A PROJECT REPORT

ON

"DETECTION OF RETINAL BLOOD VESSEL USING DEEP LEARNNG"

Submitted to

UNIVERSITY OF MUMBAI

In Partial Fulfilment of the Requirement for the Award of
BACHELOR'S DEGREE IN
COMPUTER ENGINEERING

BY

Khan Sahil Jafar Reshma	**16DCO58**
Shaikh Mohammed Yusuf Abdul Salam Mumtaz	**16DCO76**
Ansari Mohd Akram Sajid Ahmed	**16DCO47**

UNDER THE GUIDANCE OF
PROF. Mubashir Khan

DEPARTMENT OF COMPUTER ENGINEERING
Anjuman-I-Islam's Kalsekar Technical Campus
SCHOOL OF ENGINEERING & TECHNOLOGY
Plot No. 2 and 3, Sector - 16, Near Thana Naka, Khandagaon, New Panvel - 410206

2018-2019

AFFILIATED TO

UNIVERSITY OF MUMBAI

A PROJECT II REPORT
ON

"DETECTION OF RETINAL BLOOD VESSEL USING DEEP LEARNING"

Submitted to
UNIVERSITY OF MUMBAI

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER ENGINEERING

BY

Khan Sahil Jafar Reshma	**16DCO58**
Shaikh Mohammed Yusuf Abdul Salam Mumtaz	**16DCO76**
Ansari Mohd Akram Sajid Ahmed Shamshad	**16DCO47**

UNDER THE GUIDANCE OF
Prof. Mubashir Khan

DEPARTMENT OF COMPUTER ENGINEERING
Anjuman-I-Islam's Kalsekar Technical Campus
SCHOOL OF ENGINEERING & TECHNOLOGY
Plot No. 2 and 3, Sector - 16, Near Thana Naka, Khandagaon, New Panvel - 410206

2018-2019
AFFILIATED TO

UNIVERSITY OF MUMBAI

Anjuman-i-Islam's Kalsekar Technical Campus

Department of Computer Engineering
SCHOOL OF ENGINEERING & TECHNOLOGY
Plot No. 2 and 3, Sector - 16, Near Thana Naka,
Khandagaon, New Panvel - 410206

CERTIFICATE

This is certify that the project entitled

"Detection Of Retinal Blood Vessel Using Deep Learning"

submitted by

Khan Sahil Jafar Reshma	**16DCO58**
Shaikh Mohammed Yusuf Abdul Salam Mumtaz	**16DCO76**
Ansari Mohd Akram Sajid Ahmed Shamshad	**16DCO47**

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Engineering) at ***Anjuman-I-Islam's Kalsekar Technical Campus, Navi Mumbai*** under the University of MUMBAI. This work is done during year 2018-2019, under our guidance.

Date: / /

(Prof. Mubashir Khan)
Project Supervisor

(Prof. Kalpana Bhodke)
Project Coordinator

(Prof. Tabrez Khan)
HOD, Computer Department

DR. ABDUL RAZAK HONNUTAGI
Director

External Examiner

Acknowledgements

We would like to take the opportunity to express our sincere thanks to our guide **Prof.Mubashir Khan**, Assistant Professor, Department of Computer Engineering, AIKTC, School of Engineering, Panvel for his invaluable support and guidance throughout my project research work. Without his kind guidance & support this was not possible.

We are grateful to him for his timely feedback which helped me track and schedule the process effectively. his time, ideas and encouragement that he gave is help us to complete our project efficiently.

We would like to express deepest appreciation towards **DR. ABDUL RAZAK HONNUTAGI**, Director, AIKTC, Navi Mumbai, **Prof.Tabrez Khan**, Head of Department of Computer Engineering and **Prof. Kalpana Bhodke**, Project Coordinator whose invaluable guidance supported us in completing this project.

At last we must express our sincere heartfelt gratitude to all the staff members of Computer Engineering Department who helped us directly or indirectly during this course of work.

Khan Sahil Jafar Reshma

Shaikh Mohammed Yusuf Abdul Salam Mumtaz

Ansari Mohd Akram Sajid Ahmed Shamshad

Project II Approval for Bachelor of Engineering

This project entitled ***Ëetection Of Retinal Blood Vessel Using Deep Leraning¨*** by ***Khan Sahil Jafar Reshma,Shaikh Mohammed Yusuf Abdul Salam Mumtaz,Ansari Mohd Akram Sajid Ahmed Shamshad*** is approved for the degree of ***Bachelor of Engineering in Department of Computer Engineering.***

Examiners

1. .....
2. .....

Supervisors

1. .....
2. .....

Chairman

.....

Declaration

We declare that this written submission represents our ideas in our own words and where others ideas or words have been included,we have adequately cited and referenced the original sources. we also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. we understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Khan Sahil Jafar Reshma
16DCO58

Shaikh Mohammed Yusuf
AbdulSalam Mumtaz
16DCO76

Ansari Mohd Akram Sajid Ahmed
16DCO47

ABSTRACT

Traditional Retinal Scans Provides color image of the scan due to which the visibility of the identification eye diseases.Thus in an effort to improve the visibility of scans we presenet our paper making use Neural Networks to get better visibility. The field of Ophthalmology has increasingly turned into medical imaging to play important role in diagnosing diseases. It requires retinal scanned images in identifi- cation of eye diseases. Determining eye disease on the basis of traditional retinal scans can sometimes be difficult due to presence of hemorrhage or thin blood vessels since the image is not very clear. Therefore this paper attempts to improve the quality of retinal scans through im- age segmentation and supervised machine learning algorithms so diagnosis can be as accurate as possible.

Keywords: Neural Networks,Opthalmology,Image Segmentation,GAN's(Generative Adversarial Network.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose

In the conventional retinal scans done by ophthalmologists image obtained is a coloured one which may hide very thin and fine blood vessels and also vessels at the edge of the image due to which the diagnosis of the patient might be effected.

In order the enhance the visibility if image and identify blood vessels we are making use of neural networks which are excellent in pattern identification task which will help doctors in getting more clarified image of the scan.

Therefore the he quality of retinal scans the reason behind it is that it helps in identification of other diseases other than any eye ailment.To consolidate everything it is beneficial for the doctors accurate diagnosis.

1.2 Project Scope

The following figure represents system architechture for our project which is mainly divided into two sides Training side and User Side. Training consists of bringing the image in suitable form and identifying features and training of neural network where as in Client side the visibility of scan will be improved based and if any image has wrongly marked blood vessels it can be reported by doctor and a feedback will be sent back to training side.

1.3 Project Goals and Objectives

1.3.1 Goals

The Goals is to Detect and Enhance the visibilty of retinal blood vessel from given input scan images of fundus using deep learning whcih gives higher accuracy in output images

1.3.2 Objectives

- To reduce the noise in the raw image.
- To preprocessed the image before input given to the model.
- To reduce false positives generated by the system.
- To tweak the neural network parameters in such a way that the result is optimized.
- To increase the visiblity of retinal blood vessels using deep learning.

1.4 Organization of Report

The remaining part of the project is organized as follows.

Chapter 2 Presents a review of related work

Chapter 3 Describes the time management and time utilization during the project

Chapter 4 introduces the Software requirement Specification of our project.

Chapter 5 Proposes the project design of the project .It represent the architectural design ,front end design and database design of the project.

Chapter 6 Presents implementation details of our project.

Chapter 7 Presents various test cases that are consider for the result.

Chapter 8 It consists of various output Screen-shot of project.

Chapter 9 Provides some concluding remarks and direction of our future work.

Chapter 2

Literature Survey

2.1 A Cross-modality Learning Approach For Vessel Segmentation in Retinal Images

The following paper uses non mydratic which allows separation of colour components.They have selected the green component due to the fact that it increases the visibility of blood vessels.After that the image is passed through a auto-encoder which removes the noise from the image.following which the image is classified for blood vessels and when found they are not stored as a whole but rather stored in patches.once the whole image is scanned the retinal vessel map is constructed and the auc curve is plot to check the accuaracy of the model.

2.1.1 Advantages of Paper

a. There is no need of pre-processing of image

2.1.2 Disadvantages of Paper

a. Requires high Resolution Images

b. Longer Training Time Required

c. More Memory Required

2.1.3 How to overcome the problems mentioned in Paper

a. The size of images in our data-set is consistent so memory requirments are not that high as compared to the this paper

2.2 Retinal Blood Vessel Segmentation Using Line Operators and Support Vector Classification

The following paper also uses non-mydriatic image in which color components can be separated and from that they use green channel in which the visibility of retinal blood vessel is increased after that the green channel is inverted channel after that the gray scale intensity of image is calculated and the point of maximum intensity is found and a line is passed through the point of maximum intensity at a particular angle and the gray scale intensity is checked through that line is gray scale value is more blood vessels are present if not the line is passed through a different angle once the suitable angle s found vessel map is constructed.

2.2.1 Advantages of Paper

a. Considering its simplicity the method provides accurate results for vessel segmentation.

2.2.2 Disadvantages of Paper

a. Some false positives are found around the border of the optic disk and in the proximity of pathological regions

b. Some problem occur also for the segmentation of the thinnest vessels or when the local contrast is quite low

2.2.3 How to overcome the problems mentioned in Paper

a. The following paper does not uses neural network therefore the accuracy is reduced our paper uses CNN due to which accuracy is maintained.

2.3 Retinal Vessel Segmentation Using Deep Neural Networks

As compared to other neural networks convolutional neural networks is used specifically for image processing which is beneficial on addition to that they have used max-pooling layer which the last layer in the neural network for classification.since it provides much better results.

2.3.1 Advantages of Paper

a. Provides results on par with state-of-art techniques

2.3.2 Disadvantages of Paper

a. More data-set required to train the neural network

b. Experimenting with the activation function could lead to better results

2.3.3 How to overcome the problems mentioned in Paper

a. Achieving same consistency as of this paper is a challenge but the training set can be reduced if the learning curve of model is reduced

2.4 Technical Review

After rigorous assessment of above research paper we found GAN's a type of neural network to be the best for this use case since it out performs other method in detection of retinal blood vessels, mainly due its U-net architecture which helps in reconstruction of image when compared to other models

2.4.1 Advantages of Technology

a. GAN's have much better accuracy when compared to other models

b. Pattern Recognition is much better due discriminator generator models

c. Lower False Positive Generated Images.

2.4.2 Reasons to use this Technology

a. Better accuracy than state-of-art algorithms.

b. Easier Reconstruction of Image due to U-net Architechture.

Chapter 3

Project Planning

3.1 Members and Capabilities

Table 3.1: Table of Capabilities

SR. No	Name of Member	Capabilities
1	Sahil Khan	Database, UI Design
2	Mohd Yusuf Shaikh	Designing and Modeling
3	Akram Ansari	Testing and Documentation

3.2 Roles and Responsibilities

Table 3.2: Table of Responsibilities

SR. No	Name of Member	Role	Responsibilities
1	Sahil Khan	Team Leader	UI Design and Integration
2	Mohd Yusuf Shaikh	Developer	Model Design
3	Akram Ansari	Developer	Testing

3.3 Assumptions and Constraints

3.3.1 Assumption

a. Our assumption is to give better visibility of blood vessel. It is helpful for doctor to detect the disease in eyes.

3.3.2 Constraints

a. The user should be able to understand how the system works.

b. The user should provide input image in proper format otherwise it will give an error and not predict the result.

3.4 Project Management Approach

We have use Waterfall model for the development of our project.In The Waterfall approach, the whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. This means that any phase in the development process begins only if the previous phase is complete. The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Analysis, Design, Construction, Testing, Production/Implementation and Maintenance.

3.5 Ground Rules for the Project

a. We intend to develop personal relationships to enhance trust and open communication

b. As team members,we will pitch in to help where necessary to help solve problem and catch-up on behind schedule work

c. When we pose an issue or a problem,we will also try to present a solution

3.6 Project Budget

The budget of the project is very low as most of the tools we have use are open source.Following are the budget for the project

a. Operating System:Linux mint(Open Source)

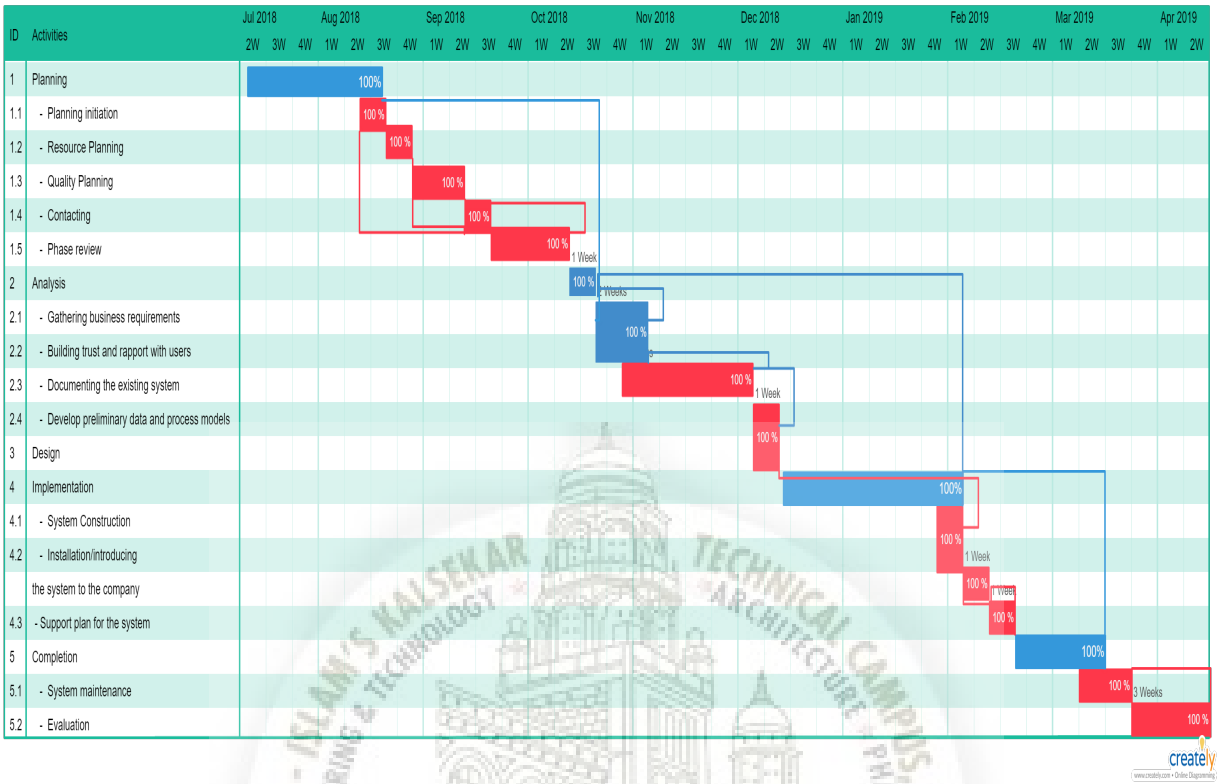
b. Google Colab for Training the model(Open Source)

c. Python programming Language for developing model

d. Tensorflow libraries

e. PHP and HTML for Web Aplication

3.7 Project Timeline



Chapter 4

Software Requirements Specification

4.1 Overall Description

4.1.1 Product Perspective

The Traditional Fundus Scanner produces images that sometimes have artifacts induced due to unusual movement of the eye or blinking of the eye. There is also a chance that some of the blood vessels might not be visible in image due to its location since the scanner can only take a two-dimensional image due to which some vessels may look faded. Due to this reason detection of eye diseases becomes difficult. Therefore, our system attempts to solve the problem using Deep Learning Models which will act as aid in detection of vessel.

4.1.2 Product Features

In this project we have mainly two major feature which is to give detection of blood vessel and enhanced the visibility of raw input images which is given to the model, As per user perspective it has to give input as a raw images of fundus and it will get the result in the highlighted format of retinal blood vessel in output display

4.1.3 User Classes and Characteristics

There is only one user (doctor) who wants to find retinal blood vessel and wants to enlighten the vessel ,As their is also facility to contribute to the dataset for making prediction better in future and making model durable for the prediction.

4.1.4 Operating Environment

Our System will provide web based application . because it is very convenient for all doctor to use from anywhere .Nowaday everything came with internet simplicity to access information from anytime anywhere.

4.1.5 Design and Implementation Constraints

The main decision was to select the neural network which is suitable for our project who give more accuracy then others and we select python lanaguage to develop the model ,about the dataset we used secondary dataset from the different website we get those dataset DRIVE and STARE are the dataset that what we are using ,how much images is given to the trained model and test model is derive by us,for web application we are using PHP and HTML for connecting web application with our model we make use of shell command in the python files.and for making model we used Tensorflow libraries and we made our function which is defined in Utils files.

4.2 System Features

This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.

4.2.1 System Feature

- Higher Accuracy
- Higher Pattern Recognition
- Lower False Positive Marking
- Higher True Positive Marking

Description and Priority

- Higher Accuracy - Refers to Prediction of blood vessels when compared to ground truth (Hand Drawn Blood Vessels).
  Priority - High
- Higher Pattern Recognition - Since the model is made of two Neural Networks it results in better pattern recognition.
  Priority - High
- Lower False Positive Marking - Due to better pattern recognition the model has lower tendency of false marking of blood vessels.
  Priority - High

- Higher True Positive Marking - Due to better pattern recognition the model has higher tendency of true marking of blood vessels.
  Priority - High

Stimulus/Response Sequences

- The User Should Log-in into the Portal and should sign to in to dashboard. If not the user should register.
- The Dashboard should show browse and upload option for image. After the image has been successfully uploaded a message should be displayed.
- After successful image upload the user should run the model for detection of blood vessels.
- The User can also contribute image to collection through contribution portal which has a similar browse and upload option if successful a message will be displayed.

Functional Requirements

REQ-1: The Lastest Browser should be installed for avoiding any compability issue
REQ-2: The User should have stable internet connection for ease of use
REQ-3: Any Operating System should work without any issue.

4.3 External Interface Requirements

4.3.1 User Interfaces

UI has one interface: Website

Website user's menu consists of

a. Home page: Here we show about project quick start
b. Login: User's login form
c. Contact us: The page which gives contact detail about us.
d. About: information about us

4.3.2 Hardware Interfaces

a. Processor: Core i3/i5/i7

b. RAM: 4GB

c. GPU: Nvdia GTX series GPU

4.3.3 Software Interfaces

a. Python for development

b. Numeric and data manipulation libraries such as

   Numpy

   Pandas

c. Machine learning Libraries such as

   Keras

   Tensorflow

d. Image processing

   SimpleITK(insight segmentation and registration toolkit)

   OpenCV

   PIL(python Imaging Library)

   Scikit-learn

4.3.4 Communications Interfaces

The system consist of three main block each of them is mandatory

a. Dataset: The dataset is intended for sorting different type of raw eye images

b. PhpMyAdmin Server: This server is intended for data management it receive the commands through API of system.

c. API: This is the core our system. By means of this function Api connects the database server (MYSQL SERVER) and generate request for data issue.

4.4 Nonfunctional Requirements

4.4.1 Performance Requirements

- Execution of model requires couple of minutes the webpage should not be interrupted during that duration.
- Load on the Server might hamper the output time required.

4.4.2 Safety Requirements

- Quality of images is the prime reason of concern since it will effect the results
- If Some user is uploading the data should verify the authenticity of the source since it will effect future results.

4.4.3 Security Requirements

- There are no such specific security requirements most of the measure are taken care of at server side.

Chapter 5

System Design

5.1 System Requirements Definition

Our System is based on detection of blood vessel using Deep learning where we are using neural network (GAN's) to train our model to predict better and provide output as clear highlighted format in image where we are giving training dataset from Drive and Stare and giving upto 40 images to train our model and after training we are testing our model and basically it produce output image with high accuracy of detecting blood vessel.

5.1.1 Functional requirements

- Higher Accuracy - Refers to Prediction of blood vessels when compared to ground truth (Hand Drawn Blood Vessels).
  Priority - High
- Higher Pattern Recognition - Since the model is made of two Neural Networks it results in better pattern recognition.
  Priority - High
- Lower False Positive Marking - Due to better pattern recognition the model has lower tendency of false marking of blood vessels.
  Priority - High
- Higher True Positive Marking - Due to better pattern recognition the model has higher tendency of true marking of blood vessels.
  Priority - High

Use-case Diagram

Description: Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. Use case diagrams are in fact two fold they are both behavior diagrams, because they describe behavior of the system, and they are also structure diagrams as a special case of class diagrams where classifiers are restricted to be either actors or use cases related to each other with associations.

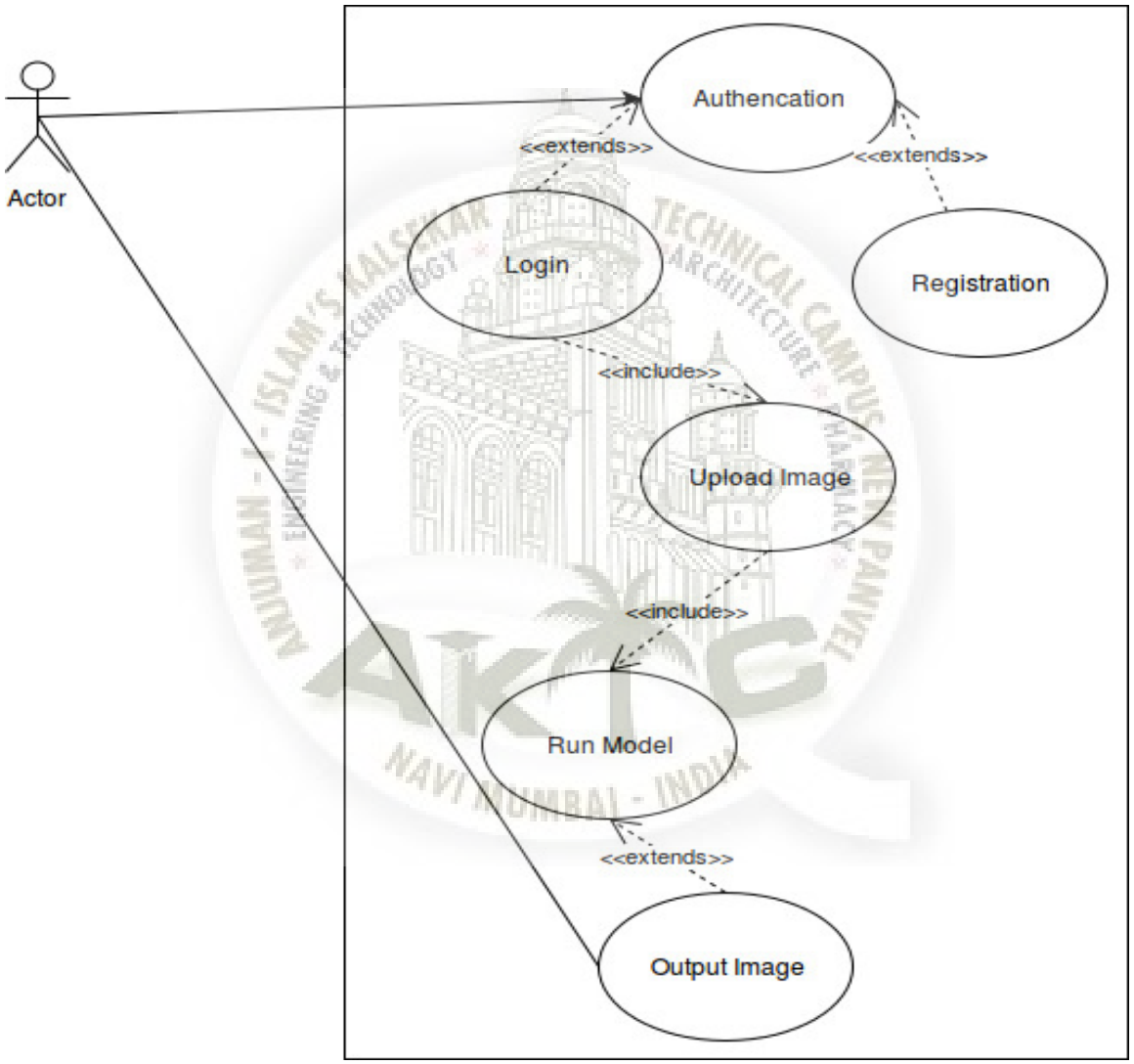


Figure 5.1: Usecase Diagram

Data-flow Diagram

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFD's that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one. Like all the best diagrams and charts, a DFD can often visually "say" things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developer to CEO. That's why DFD's remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time or database-oriented software or systems.

Level 0: The first diagram is the level 0 Data flow diagram of the system in which, the modules, which will be there after the deployment are shown. DFD Level 0 It contains total number of process(1) in our DFD level 0 diagram.where doctor interact with the system.

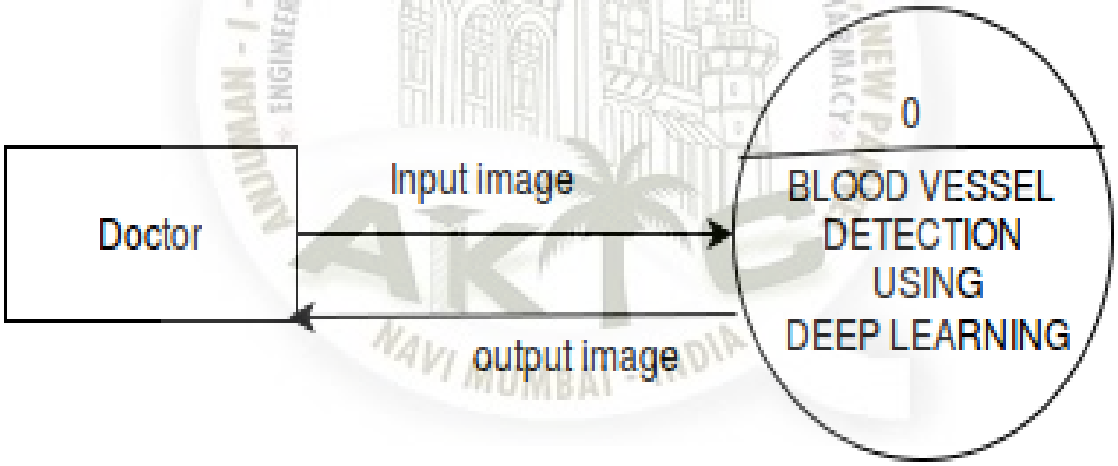


Figure 5.2: Level 0 DFD Diagram

Level 1: The second diagram is the level 1 Data Flow Diagram of the system in which the modules are briefly explained with their functionalities.

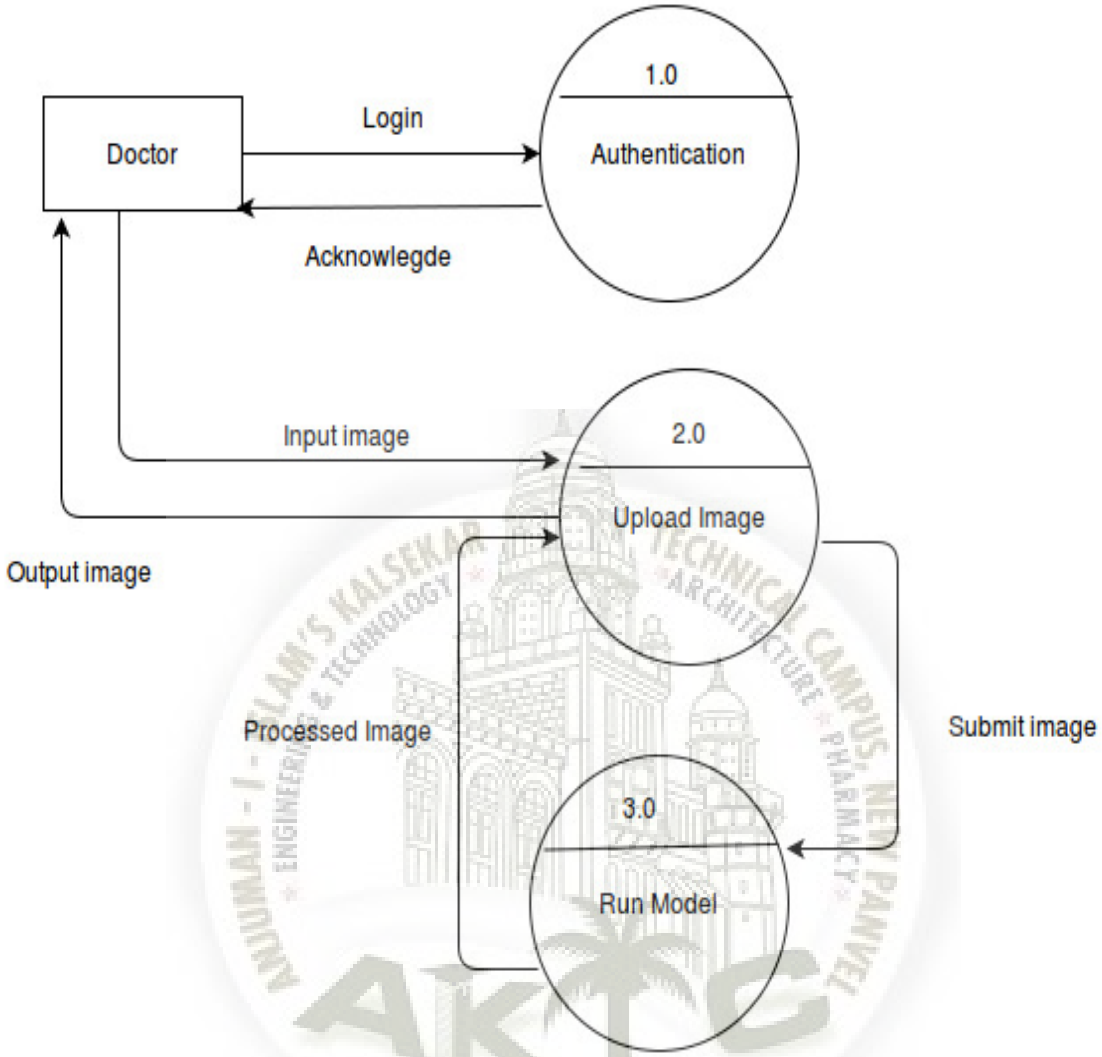


Figure 5.3: Level 1 DFD Diagram

5.1.2 System requirements (non-functional requirements)

a. Usability requirement-Giving the better visibility of blood vessel.it is helpful for doctor to detect the disease from that blood vessel

b. Efficiency requirement- The dataset should be able to response quickly when doctor upload the raw image.

c. Performance requirement-The website should be able to response the queries submitted by the user without delay.

d. Reliability requirement- The website should work under all conditions and performed the required functionality

Database Schema/ E-R Diagram

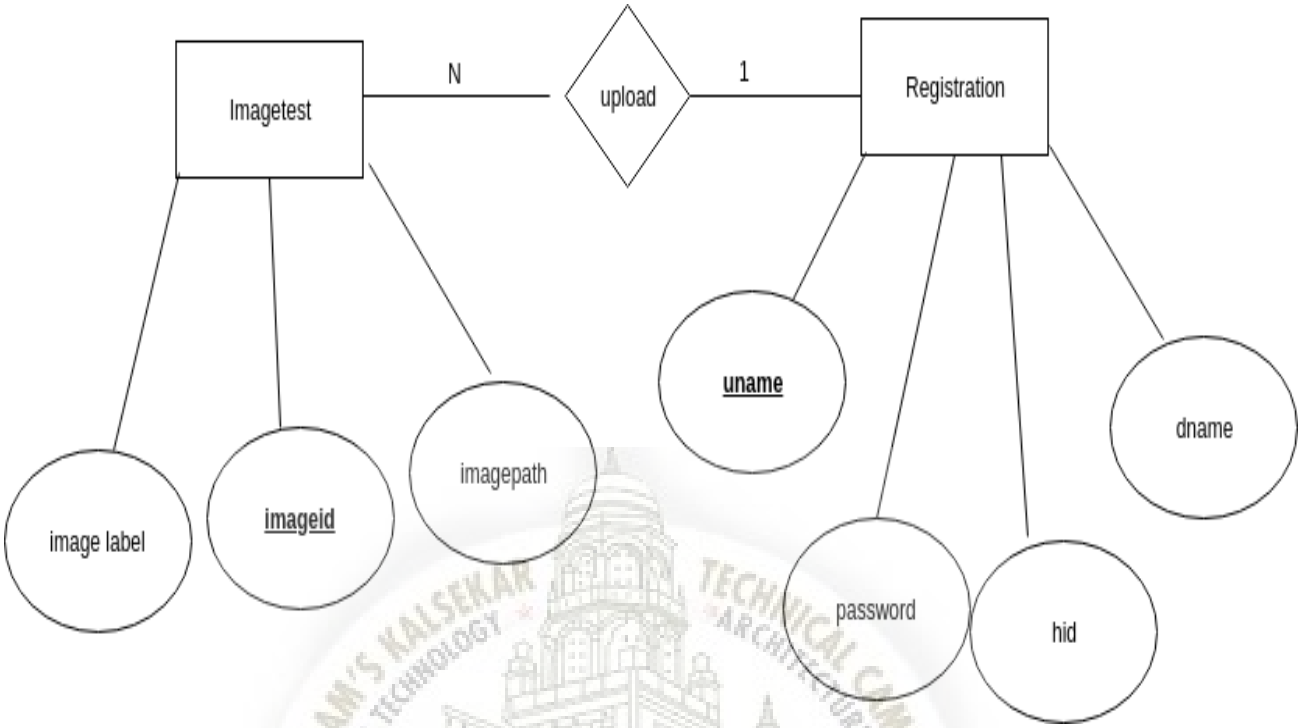


Figure 5.4: ER Diagram

Figure above illustrates the ER diagram of our system. The ER or (Entity Relational Model) is a high-level conceptual data model diagram. Entity-Relation model is based on the notion of real-world entities and the relationship between them. ER modeling helps us to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing our database.

5.2 System Architecture Design

The following figure represents system architecture for our project which is mainly divided into two sides Training side and User Side. Training consists of bringing the image in suitable form and identifying features and training of neural network where as in Client side the visibility of scan will be improved based and if any image has wrongly marked blood vessels it can be reported by doctor and a feedback will be sent back to training side.

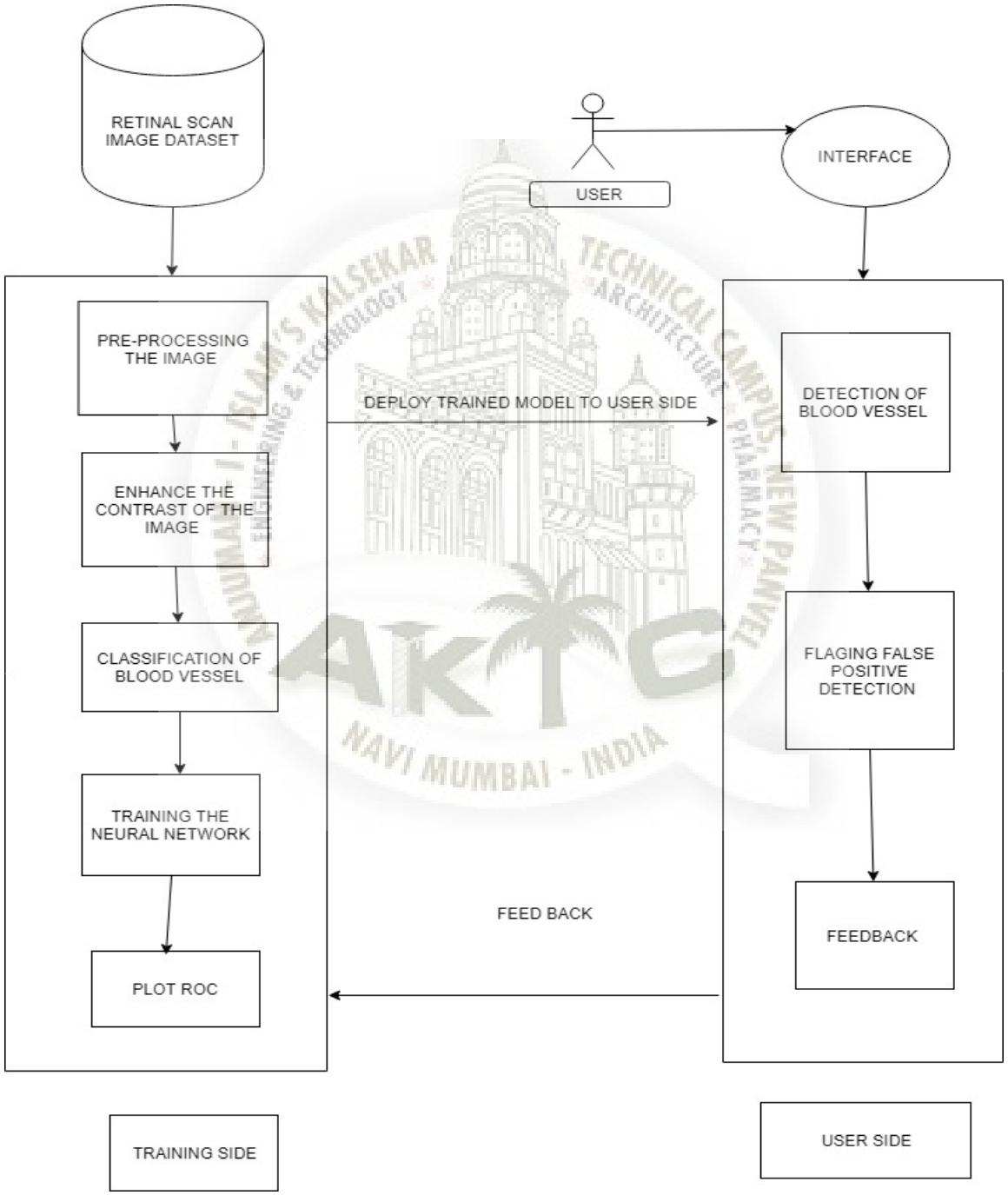


Figure 5.5: System Architecture

5.3 Sub-system Development

Here we discuss each of the project modules which are Dataset, Model, Solver, Tensorflowutils, evaluation, main and utils.

5.3.1 Dataset Module

The task of this module is to only fetch or read images from the dataset located in the appropriate directory. There are two dataset to fetch from DRIVE and STARE.

5.3.2 Model Module

This is the most crucial module of the project where the Neural Network is build. Our Model consists of two neural networks Discriminator and Generator both of which are defined here. The Primary task is extraction of features using CGAN and reconstruct the image using the feature Vector. The module also comprises of function which are used to train the neural network.

5.3.3 Evaluation Module

To test the Model evaluation module is used. It Inputs the Model the test dataset and then based on the model training it predicts the blood vessel. The prediction of blood vessel is done using various methods to provide a brief comparison of how well it performs model performs and accuracy of model is also measured on various parameters.

5.3.4 Main Module

This module provides the initialization parameters for training the model and checks for certain pre-requisties.

5.3.5 Solver Module

This module is called after the training of the Model for showing statistics about what accuracy has been achieved During the training. If model is trained on various dataset and particular model shows hihger rate of accuracy then that model weights can be saved for further use.

5.3.6 Tensorflowutils Module

This module has rewritten library function for ease of use according to the requirements.

5.3.7 utils Module

The module is toolbox for other modules as all the accessory functions are written here such as performing operations on the dataset, fetching images from the dataset, Calculation of various accuracy of parameters are done here such as Sensitivity, Specificity, f1 score , Precision Recall, dice coefficient.

5.4 Systems Integration

To Fully integrate system all required dependency has to be satisfied to be such as libraries needed which are Tensorflow, PIL, OS, Solver, TensorFlowutils etc and the dataset for performing testing.

5.4.1 Class Diagram

This is the Class diagram of the system in which the modules which will be there after the deployment are shown. This class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity

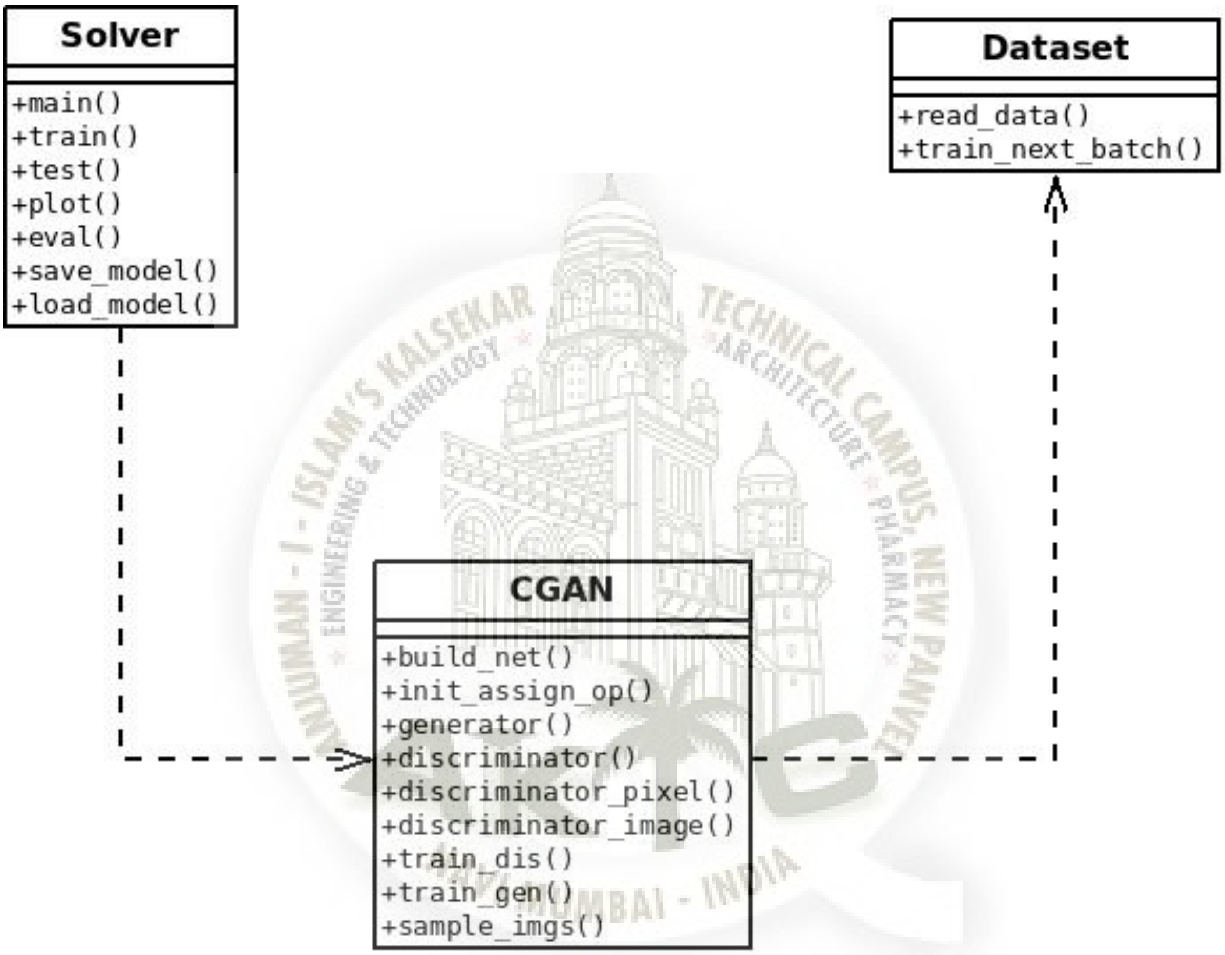


Figure 5.6: Class Diagram

5.4.2 Sequence Diagram

To understand what a sequence diagram is, it's important to know the role of UML, or the Unified Modeling Language, is a modeling toolkit that guides the creation and notation of many types of diagrams, including behavior diagrams, interaction diagrams, and structure diagrams. Sequence diagrams are a kind of interaction diagram, because they describe how and in what order a group of objects works together. These diagrams are used by software developers and business people alike to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

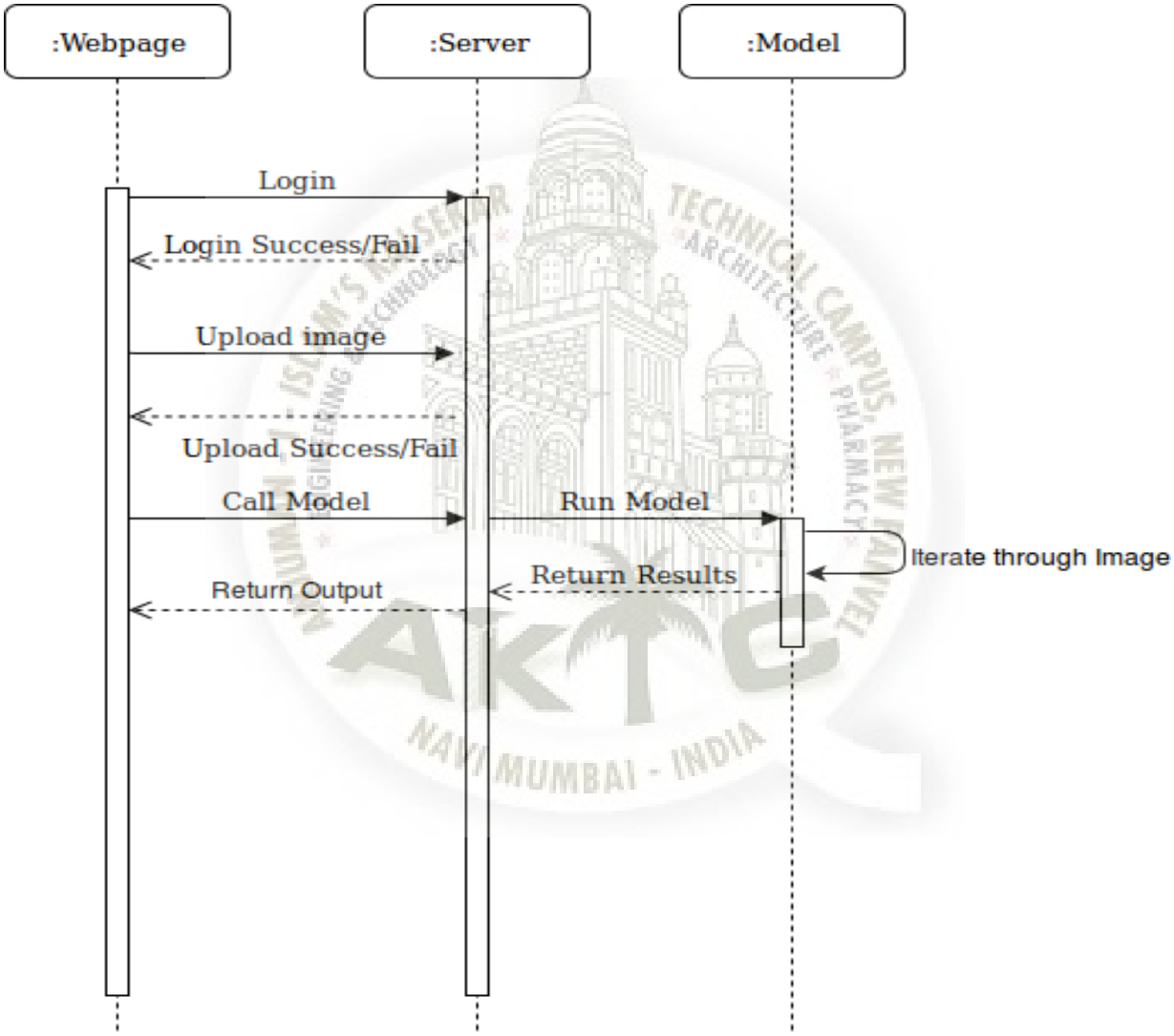


Figure 5.7: Sequence Diagram

5.4.3 Component Diagram

Component diagram are different in terms of nature and behaviour .Component diagrams are used to mode the physical aspects of system.Now the question is,what are these physical aspects? Physical aspects are the element such as executable,libraries ,files,documents ,etc. which reside in a node .Component diagram are used to visualize the organization and relationship among component in a system .these diagram are also used to make executable system.

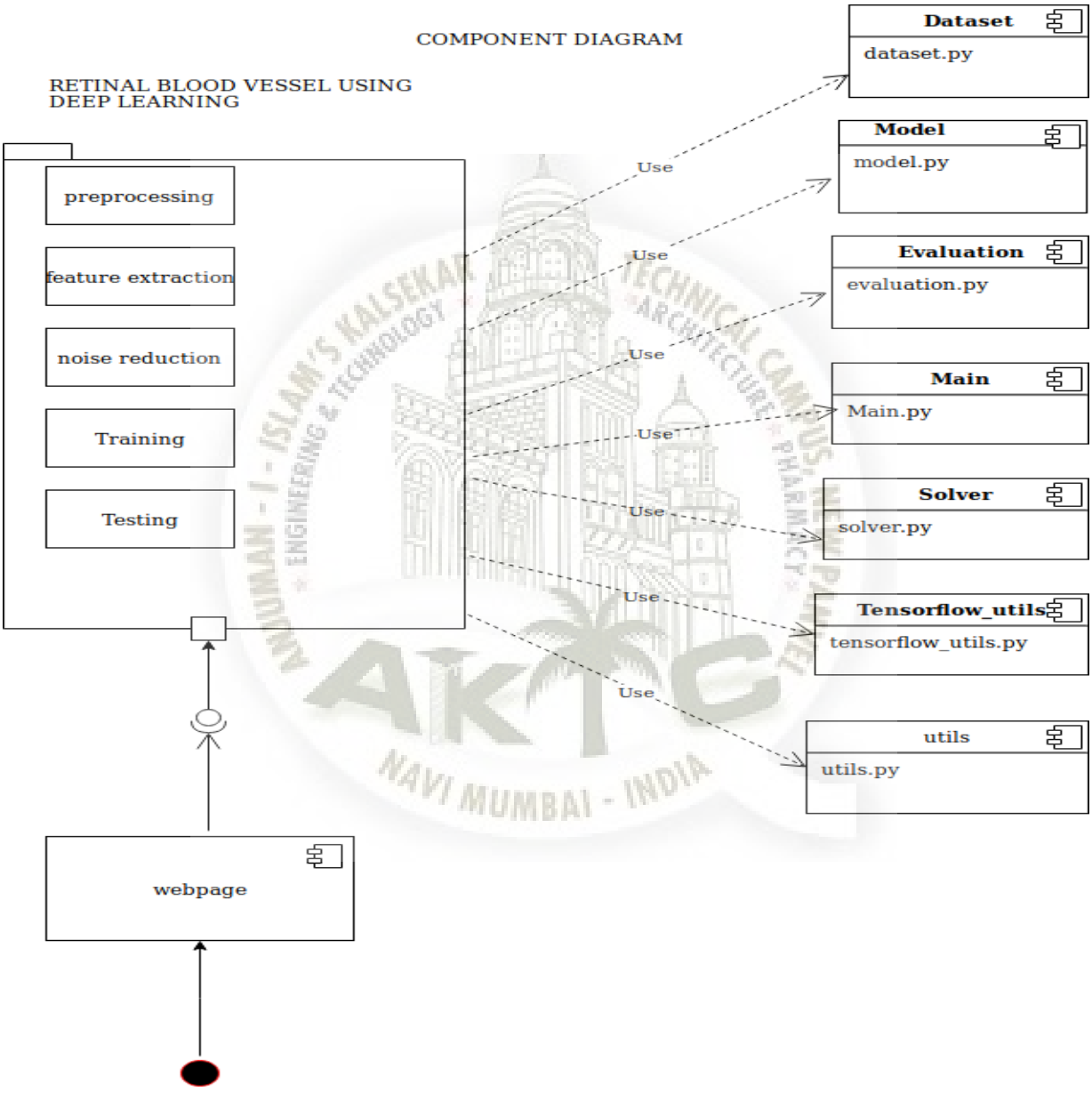


Figure 5.8: Component Diagram

5.4.4 Deployment Diagram

Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets. The below figure describe the Deployment Diagram of our system.

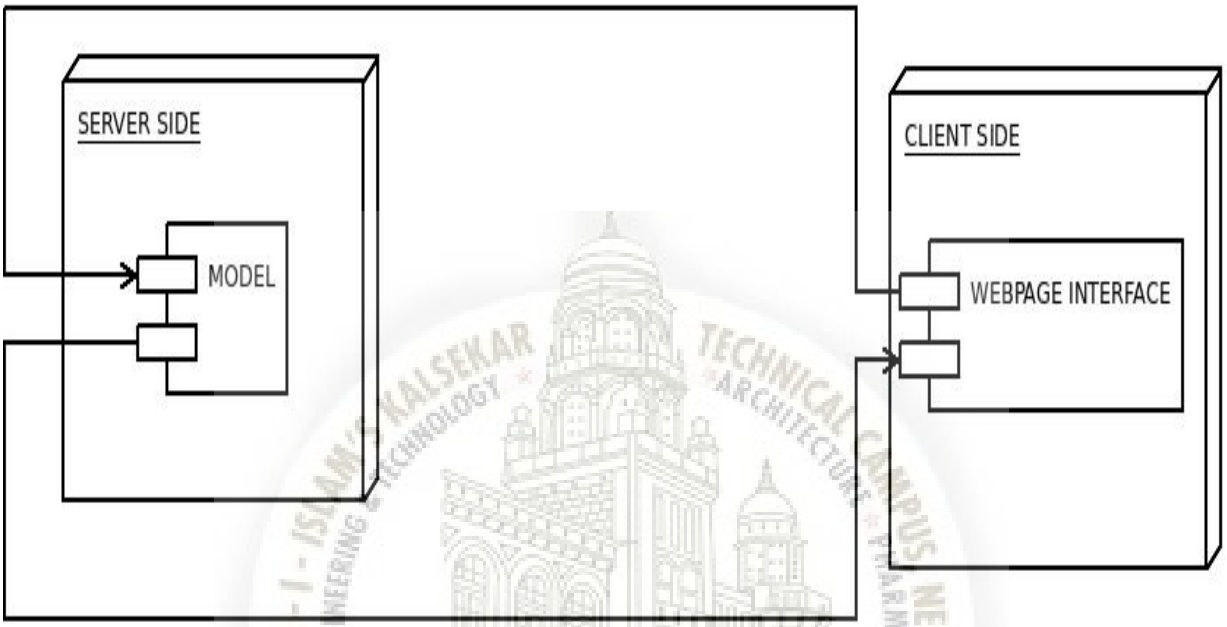


Figure 5.9: Deployment Diagram

Chapter 6

Implementation

6.1 Dataset

The task of this module is to only fetch or read images from the dataset located in the appropriate directory. There are two datasets to fetch from DRIVE and STARE

```
import os
import random
import numpy as np
from datetime import datetime

import utils as utils


class Dataset(object):
    def __init__(self, dataset, flags):
        self.dataset = dataset
        self.flags = flags

        self.image_size = (640, 640) if self.dataset == 'DRIVE' else (720, 720)
        self.ori_shape = (584, 565) if self.dataset == 'DRIVE' else (605, 700)
        self.val_ratio = 0.1  # 10% of the training data are used as validation
            data
        self.train_dir = "../data/{}/training/".format(self.dataset)
        self.test_dir = "../data/{}/test/".format(self.dataset)

        self.num_train, self.num_val, self.num_test = 0, 0, 0

        self._read_data()  # read training, validation, and test data)
        print('num of training images: {}'.format(self.num_train))
        print('num of validation images: {}'.format(self.num_val))
        print('num of test images: {}'.format(self.num_test))

    def _read_data(self):
        if self.flags.is_test:
            # real test images and vessels in the memory
            self.test_imgs, self.test_vessels, self.test_masks, self.
                test_mean_std = utils.get_test_imgs(
                 target_dir=self.test_dir, img_size=self.image_size, dataset=self
                    .dataset)
            self.test_img_files = utils.all_files_under(os.path.join(self.
                test_dir, 'images'))

            self.num_test = self.test_imgs.shape[0]

```

```
        elif not self.flags.is_test:
            random.seed(datetime.now())  # set random seed
            self.train_img_files, self.train_vessel_files, mask_files = utils.
                get_img_path(
                 self.train_dir, self.dataset)

            self.num_train = int(len(self.train_img_files))
            self.num_val = int(np.floor(self.val_ratio * int(len(self.
                train_img_files))))
            self.num_train -= self.num_val

            self.val_img_files = self.train_img_files[-self.num_val:]
            self.val_vessel_files = self.train_vessel_files[-self.num_val:]
            val_mask_files = mask_files[-self.num_val:]
            self.train_img_files = self.train_img_files[:-self.num_val]
            self.train_vessel_files = self.train_vessel_files[:-self.num_val]

            # read val images and vessels in the memory
            self.val_imgs, self.val_vessels, self.val_masks, self.val_mean_std =
                 utils.get_val_imgs(
                 self.val_img_files, self.val_vessel_files, val_mask_files,
                    img_size=self.image_size)

            self.num_val = self.val_imgs.shape[0]

    def train_next_batch(self, batch_size):
        train_indices = np.random.choice(self.num_train, batch_size, replace=
            True)
        train_imgs, train_vessels = utils.get_train_batch(
            self.train_img_files, self.train_vessel_files, train_indices.astype(
                np.int32),
            img_size=self.image_size)
        train_vessels = np.expand_dims(train_vessels, axis=3)

        return train_imgs, train_vessels
```

6.2 Evaluation

To test the Model evaluation module is used. It Inputs the Model the test dataset and then based on the model training it predicts the blood vessel. The prediction of blood vessel is done using various methods to provide a brief comparison of how well it performs model performs and accuracy of model is also measured on various parameters.

```
 #!/usr/bin/env python3
print("starting...")
import os
import numpy as np
from PIL import Image
from sklearn.metrics import roc_curve, precision_recall_curve
from sklearn.metrics.classification import confusion_matrix

import utils

# set output directories
comparison_out = "../evaluation/{}/comparison/{}"
vessels_out = "../evaluation/{}/vessels/{}"
curves_out = "../evaluation/{}/measures".
testdata = "../data/{}/test/images"
print("check")
# draw
result_dir = "../results"
datasets = utils.all_files_under(result_dir)

for dataset in datasets:
    print('<=== {} ===>\n'.format(os.path.basename(dataset)))

    all_results = utils.all_files_under(dataset)
    # mask
    mask_dir = os.path.join(dataset, "mask")
    masks = utils.load_images_under_dir(mask_dir) / 255
    # gt vessel
    gt_dir = os.path.join(dataset, "1st_manual")
    gt_vessels = utils.load_images_under_dir(gt_dir) / 255

    # collect results from all methods
    methods = []
    fprs, tprs, precs, recalls = [], [], [], []
    human_op_pts_roc, human_op_pts_pr = None, None
    for result in all_results:
        if "mask" not in result:  # skip mask and ground truth
            # get pixels inside the field of view in fundus images
            pred_vessels = utils.load_images_under_dir(result) / 255
            gt_vessels_in_mask, pred_vessels_in_mask = utils.
                pixel_values_in_mask(
                gt_vessels, pred_vessels, masks)

            # visualize results
            if "V-GAN" in result or "DRIU" in result or "1st_manual" in result:
                test_dir = testdata.format(os.path.basename(dataset))
                ori_imgs = utils.load_images_under_dir(test_dir)
                vessels_dir = vessels_out.format(os.path.basename(dataset), os.
                    path.basename(result))
                filenames = utils.all_files_under(result)
                if not os.path.isdir(vessels_dir):
```

```
                os.makedirs(vessels_dir)

            for index in range(gt_vessels.shape[0]):

                thresholded_vessel = utils.threshold_by_otsu(
                    np.expand_dims(pred_vessels[index, ...], axis=0),
                    np.expand_dims(masks[index, ...], axis=0), flatten=False
                        )*255

                ori_imgs[index, ...][np.squeeze(thresholded_vessel, axis=0)
                    == 0] = (0, 0, 0)

                Image.fromarray(ori_imgs[index, ...].astype(np.uint8)).save(
                    os.path.join(vessels_dir, os.path.basename(filenames[
                        index])))

            # compare with the ground truth
            comp_dir = comparison_out.format(os.path.basename(dataset), os.
                path.basename(result))
            if not os.path.isdir(comp_dir):
                os.makedirs(comp_dir)

            dice_list = []
            for index in range(gt_vessels.shape[0]):
                diff_map, dice_coeff = utils.difference_map(gt_vessels[index
                    , ...],
                                                            pred_vessels[
                                                                index, ...],
                                                            masks[index,
                                                                ...])
                dice_list.append(dice_coeff)
                Image.fromarray(diff_map.astype(np.uint8)).save(
                    os.path.join(comp_dir, os.path.basename(filenames[index
                        ])))

            # print("indices of best dice coeff : {}".format(sorted(range(
                len(dice_list)),
            #                                                    key=
                lambda k: dice_list[k])))

        # skip the ground truth
        if "1st_manual" not in result:
            # print metrics
            print("-- {} --".format(os.path.basename(result)))
            print("dice coefficient : {:.4f}".format(
                utils.dice_coefficient(gt_vessels, pred_vessels, masks)))
            print("f1 score : {:.4f},\naccuracy : {:.4f},\nsensitivity :
                {:.4f},\nspecificity : {:.4f}\n"
                  .format(*utils.misc_measures_evaluation(gt_vessels,
                      pred_vessels, masks)))

            # compute false positive rate, true positive graph
            method = os.path.basename(result)
            methods.append(method)

            if method == 'CRFs' or method == '2nd_manual':
                cm = confusion_matrix(gt_vessels_in_mask,
                    pred_vessels_in_mask)
                fpr = 1 - 1. * cm[0, 0] / (cm[0, 1] + cm[0, 0])
                tpr = 1. * cm[1, 1] / (cm[1, 0] + cm[1, 1])
                prec = 1. * cm[1, 1] / (cm[0, 1] + cm[1, 1])
```

```
                    recall = tpr

                    if method == '2nd_manual':
                        human_op_pts_roc, human_op_pts_pr = utils.
                            operating_pts_human_experts(
                            gt_vessels, pred_vessels, masks)
                else:
                    fpr, tpr, _ = roc_curve(gt_vessels_in_mask,
                        pred_vessels_in_mask)
                    prec, recall, _ = precision_recall_curve(gt_vessels_in_mask,
                        pred_vessels_in_mask)

                fprs.append(fpr)
                tprs.append(tpr)
                precs.append(prec)
                recalls.append(recall)

    # save plots of ROC and PR curves
    curve_dir = curves_out.format(os.path.basename(dataset))
    if not os.path.isdir(curve_dir):
        os.makedirs(curve_dir)

    utils.plot_AUC_ROC(fprs, tprs, methods, curve_dir, human_op_pts_roc)
    utils.plot_AUC_PR(precs, recalls, methods, curve_dir, human_op_pts_pr)
```

6.3 Main

This module provides the initialization parameters for training the model and checks for certain pre-requisties.

```

import os
import tensorflow as tf
from solver import Solver

FLAGS = tf.flags.FLAGS

tf.flags.DEFINE_integer('train_interval', 1, 'training interval between
    discriminator and generator, default: 1')
tf.flags.DEFINE_integer('ratio_gan2seg', 10, 'ratio of gan loss to seg loss,
    default: 10')
tf.flags.DEFINE_string('gpu_index', '0', 'gpu index, default: 0')
tf.flags.DEFINE_string('discriminator', 'image', 'type of discriminator [pixel|
    patch1|patch2|image], '
                                                 'default: image')

tf.flags.DEFINE_integer('batch_size', 1, 'batch size, default: 1')
tf.flags.DEFINE_string('dataset', 'STARE', 'dataset name [DRIVE|STARE], default:
     STARE')
tf.flags.DEFINE_bool('is_test', False, 'default: False (train)')

tf.flags.DEFINE_float('learning_rate', 2e-4, 'initial learning rate for Adam,
    default: 2e-4')
tf.flags.DEFINE_float('beta1', 0.5, 'momentum term of adam, default: 0.5')
tf.flags.DEFINE_integer('iters', 50000, 'number of iteratons, default: 50000')
tf.flags.DEFINE_integer('print_freq', 100, 'print frequency, default: 100')
tf.flags.DEFINE_integer('eval_freq', 500, 'evaluation frequency, default: 500')
tf.flags.DEFINE_integer('sample_freq', 200, 'sample frequency, default: 200')

tf.flags.DEFINE_string('checkpoint_dir', './checkpoints', 'models are saved here
    ')
tf.flags.DEFINE_string('sample_dir', './sample', 'sample are saved here')
tf.flags.DEFINE_string('test_dir', './test', 'test images are saved here')


def main(_):
    os.environ['CUDA_VISIBLE_DEVICES'] = FLAGS.gpu_index

    solver = Solver(FLAGS)
    if FLAGS.is_test:
        solver.test()
    if not FLAGS.is_test:
        solver.train()


if __name__ == '__main__':
    tf.app.run()
```

6.4 Solver

This module is called after the training of the Model for showing statistics about what accuracy has been achieved During the training. If model is trained on various datasets and particular model shows higer rate of accuarcy then that model weights can be saved for further use.

```
1
2  import os
3  import time
4  import collections
5  import numpy as np
6  import matplotlib.pyplot as plt
7  import matplotlib.gridspec as gridspec
8  import tensorflow as tf
9  from PIL import Image
10
11 from dataset import Dataset
12 # noinspection PyPep8Naming
13 import TensorFlow_utils as tf_utils
14 import utils as utils
15 from model import CGAN
16
17
18 class Solver(object):
19     def __init__(self, flags):
20         run_config = tf.ConfigProto()
21         run_config.gpu_options.allow_growth = True
22         self.sess = tf.Session(config=run_config)
23
24         self.flags = flags
25         self.dataset = Dataset(self.flags.dataset, self.flags)
26         self.model = CGAN(self.sess, self.flags, self.dataset.image_size)
27
28         self.best_auc_sum = 0.
29         self._make_folders()
30
31         self.saver = tf.train.Saver()
32         self.sess.run(tf.global_variables_initializer())
33
34         tf_utils.show_all_variables()
35
36     def _make_folders(self):
37         self.model_out_dir = "{}/model_{}_{}_{}".format(self.flags.dataset, self
               .flags.discriminator,
38                                                         self.flags.
                                                            train_interval, self
                                                            .flags.batch_size)
39         if not os.path.isdir(self.model_out_dir):
40             os.makedirs(self.model_out_dir)
41
42         if self.flags.is_test:
43             self.img_out_dir = "{}/seg_result_{}_{}_{}".format(self.flags.
                   dataset,
44                                                                self.flags.
                                                                   discriminator
                                                                   ,
45                                                                self.flags.
                                                                   train_interval
                                                                   ,
```

```
                                                                    self.flags.
                                                                        batch_size)
            self.auc_out_dir = "{}/auc_{}_{}_{}".format(self.flags.dataset, self
                .flags.discriminator,
                                                        self.flags.
                                                            train_interval, self
                                                            .flags.batch_size)

            if not os.path.isdir(self.img_out_dir):
                os.makedirs(self.img_out_dir)
            if not os.path.isdir(self.auc_out_dir):
                os.makedirs(self.auc_out_dir)

        elif not self.flags.is_test:
            self.sample_out_dir = "{}/sample_{}_{}_{}".format(self.flags.dataset
                , self.flags.discriminator,
                                                              self.flags.
                                                                  train_interval
                                                                  , self.flags.
                                                                  batch_size)
            if not os.path.isdir(self.sample_out_dir):
                os.makedirs(self.sample_out_dir)

    def train(self):
        for iter_time in range(0, self.flags.iters+1, self.flags.train_interval)
            :
            self.sample(iter_time)  # sampling images and save them

            # train discrminator
            for iter_ in range(1, self.flags.train_interval+1):
                x_imgs, y_imgs = self.dataset.train_next_batch(batch_size=self.
                    flags.batch_size)
                d_loss = self.model.train_dis(x_imgs, y_imgs)
                self.print_info(iter_time + iter_, 'd_loss', d_loss)

            # train generator
            for iter_ in range(1, self.flags.train_interval+1):
                x_imgs, y_imgs = self.dataset.train_next_batch(batch_size=self.
                    flags.batch_size)
                g_loss = self.model.train_gen(x_imgs, y_imgs)
                self.print_info(iter_time + iter_, 'g_loss', g_loss)

            auc_sum = self.eval(iter_time, phase='train')

            if self.best_auc_sum < auc_sum:
                self.best_auc_sum = auc_sum
                self.save_model(iter_time)

    def test(self):
        if self.load_model():
            print(' [*] Load Success!\n')
            self.eval(phase='test')
        else:
            print(' [!] Load Failed!\n')

    def sample(self, iter_time):
        if np.mod(iter_time, self.flags.sample_freq) == 0:
            idx = np.random.choice(self.dataset.num_val, 2, replace=False)
            x_imgs, y_imgs = self.dataset.val_imgs[idx], self.dataset.
                val_vessels[idx]
            samples = self.model.sample_imgs(x_imgs)
```

```
95
96              # masking
97              seg_samples = utils.remain_in_mask(samples, self.dataset.val_masks[
                    idx])
98
99              # crop to original image shape
100             x_imgs_ = utils.crop_to_original(x_imgs, self.dataset.ori_shape)
101             seg_samples_ = utils.crop_to_original(seg_samples, self.dataset.
                    ori_shape)
102             y_imgs_ = utils.crop_to_original(y_imgs, self.dataset.ori_shape)
103
104             # sampling
105             self.plot(x_imgs_, seg_samples_, y_imgs_, iter_time, idx=idx,
                    save_file=self.sample_out_dir,
106                       phase='train')
107
108     def plot(self, x_imgs, samples, y_imgs, iter_time, idx=None, save_file=None,
            phase='train'):
109         # initialize grid size
110         cell_size_h, cell_size_w = self.dataset.ori_shape[0] / 100, self.dataset
                .ori_shape[1] / 100
111         num_columns, margin = 3, 0.05
112         width = cell_size_w * num_columns
113         height = cell_size_h * x_imgs.shape[0]
114         fig = plt.figure(figsize=(width, height))  # (column, row)
115         gs = gridspec.GridSpec(x_imgs.shape[0], num_columns)  # (row, column)
116         gs.update(wspace=margin, hspace=margin)
117
118         # convert from normalized to original image
119         x_imgs_norm = np.zeros_like(x_imgs)
120         std, mean = 0., 0.
121         for _ in range(x_imgs.shape[0]):
122             if phase == 'train':
123                 std = self.dataset.val_mean_std[idx[_]]['std']
124                 mean = self.dataset.val_mean_std[idx[_]]['mean']
125             elif phase == 'test':
126                 std = self.dataset.test_mean_std[idx[_]]['std']
127                 mean = self.dataset.test_mean_std[idx[_]]['mean']
128             x_imgs_norm[_] = np.expand_dims(x_imgs[_], axis=0) * std + mean
129         x_imgs_norm = x_imgs_norm.astype(np.uint8)
130
131         # 1 channel to 3 channels
132         samples_3 = np.stack((samples, samples, samples), axis=3)
133         y_imgs_3 = np.stack((y_imgs, y_imgs, y_imgs), axis=3)
134
135         imgs = [x_imgs_norm, samples_3, y_imgs_3]
136         for col_index in range(len(imgs)):
137             for row_index in range(x_imgs.shape[0]):
138                 ax = plt.subplot(gs[row_index * num_columns + col_index])
139                 plt.axis('off')
140                 ax.set_xticklabels([])
141                 ax.set_yticklabels([])
142                 ax.set_aspect('equal')
143                 plt.imshow(imgs[col_index][row_index].reshape(
144                     self.dataset.ori_shape[0], self.dataset.ori_shape[1], 3),
                            cmap='Greys_r')
145
146         if phase == 'train':
147             plt.savefig(save_file + '/{}_{}.png'.format(str(iter_time), idx[0]),
                        bbox_inches='tight')
148             plt.close(fig)
```

```
149         else:
150             # save compared image
151             plt.savefig(os.path.join(save_file, 'compared_{}.png'.format(os.path
                    .basename(
152                 self.dataset.test_img_files[idx[0]])[:-4])), bbox_inches='tight'
                        )
153             plt.close(fig)
154 
155             # save vessel alone, vessel should be uint8 type
156             Image.fromarray(np.squeeze(samples*255).astype(np.uint8)).save(os.
                    path.join(
157                 save_file, '{}.png'.format(os.path.basename(self.dataset.
                        test_img_files[idx[0]][:-4]))))
158 
159     def print_info(self, iter_time, name, loss):
160         if np.mod(iter_time, self.flags.print_freq) == 0:
161             ord_output = collections.OrderedDict([(name, loss), ('dataset', self
                    .flags.dataset),
162                                                   ('discriminator', self.flags.
                                                       discriminator),
163                                                   ('train_interval', np.float32(
                                                       self.flags.train_interval)
                                                       ),
164                                                   ('gpu_index', self.flags.
                                                       gpu_index)])
165             utils.print_metrics(iter_time, ord_output)
166 
167     def eval(self, iter_time=0, phase='train'):
168         total_time, auc_sum = 0., 0.
169         if np.mod(iter_time, self.flags.eval_freq) == 0:
170             num_data, imgs, vessels, masks = None, None, None, None
171             if phase == 'train':
172                 num_data = self.dataset.num_val
173                 imgs = self.dataset.val_imgs
174                 vessels = self.dataset.val_vessels
175                 masks = self.dataset.val_masks
176             elif phase == 'test':
177                 num_data = self.dataset.num_test
178                 imgs = self.dataset.test_imgs
179                 vessels = self.dataset.test_vessels
180                 masks = self.dataset.test_masks
181 
182             generated = []
183             for iter_ in range(num_data):
184                 x_img = imgs[iter_]
185                 x_img = np.expand_dims(x_img, axis=0)  # (H, W, C) to (1, H, W,
                        C)
186 
187                 # measure inference time
188                 start_time = time.time()
189                 generated_vessel = self.model.sample_imgs(x_img)
190                 total_time += (time.time() - start_time)
191 
192                 generated.append(np.squeeze(generated_vessel, axis=(0, 3)))  #
                        (1, H, W, 1) to (H, W)
193 
194             generated = np.asarray(generated)
195             # calculate measurements
196             auc_sum = self.measure(generated, vessels, masks, num_data,
                    iter_time, phase, total_time)
197 
```

```
198             if phase == 'test':
199                 # save test images
200                 segmented_vessel = utils.remain_in_mask(generated, masks)
201 
202                 # crop to original image shape
203                 imgs_ = utils.crop_to_original(imgs, self.dataset.ori_shape)
204                 cropped_vessel = utils.crop_to_original(segmented_vessel, self.
                        dataset.ori_shape)
205                 vessels_ = utils.crop_to_original(vessels, self.dataset.
                        ori_shape)
206 
207                 for idx in range(num_data):
208                     self.plot(np.expand_dims(imgs_[idx], axis=0),
209                               np.expand_dims(cropped_vessel[idx], axis=0),
210                               np.expand_dims(vessels_[idx], axis=0),
211                               'test', idx=[idx], save_file=self.img_out_dir,
                                  phase='test')
212 
213         return auc_sum
214 
215     def measure(self, generated, vessels, masks, num_data, iter_time, phase,
            total_time):
216         # masking
217         vessels_in_mask, generated_in_mask = utils.pixel_values_in_mask(
218             vessels, generated, masks)
219 
220         # averaging processing time
221         avg_pt = (total_time / num_data) * 1000  # average processing tiem
222 
223         # evaluate Area Under the Curve of ROC and Precision-Recall
224         auc_roc = utils.AUC_ROC(vessels_in_mask, generated_in_mask)
225         auc_pr = utils.AUC_PR(vessels_in_mask, generated_in_mask)
226 
227         # binarize to calculate Dice Coeffient
228         binarys_in_mask = utils.threshold_by_otsu(generated, masks)
229         dice_coeff = utils.dice_coefficient_in_train(vessels_in_mask,
                binarys_in_mask)
230         acc, sensitivity, specificity = utils.misc_measures(vessels_in_mask,
                binarys_in_mask)
231         score = auc_pr + auc_roc + dice_coeff + acc + sensitivity + specificity
232 
233         # auc_sum for saving best model in training
234         auc_sum = auc_roc + auc_pr
235 
236         # print information
237         ord_output = collections.OrderedDict([('auc_pr', auc_pr), ('auc_roc',
                auc_roc),
238                                               ('dice_coeff', dice_coeff), ('acc'
                                                  , acc),
239                                               ('sensitivity', sensitivity), ('
                                                  specificity', specificity),
240                                               ('score', score), ('auc_sum',
                                                  auc_sum),
241                                               ('best_auc_sum', self.best_auc_sum
                                                  ), ('avg_pt', avg_pt)])
242         utils.print_metrics(iter_time, ord_output)
243 
244         # write in tensorboard when in train mode only
245         if phase == 'train':
246             self.model.measure_assign(
247                 auc_pr, auc_roc, dice_coeff, acc, sensitivity, specificity,
```

```
                    score, iter_time)
        elif phase == 'test':
            # write in npy format for evaluation
            utils.save_obj(vessels_in_mask, generated_in_mask,
                           os.path.join(self.auc_out_dir, "auc_roc.npy"),
                           os.path.join(self.auc_out_dir, "auc_pr.npy"))

        return auc_sum

    def save_model(self, iter_time):
        self.model.best_auc_sum_assign(self.best_auc_sum)

        model_name = "iter_{}_auc_sum_{:.3}".format(iter_time, self.best_auc_sum
            )
        self.saver.save(self.sess, os.path.join(self.model_out_dir, model_name))

        print('=====================================================')
        print('                      Model saved!                   ')
        print(' Best auc_sum: {:.3}'.format(self.best_auc_sum))
        print('=====================================================\n')

    def load_model(self):
        print(' [*] Reading checkpoint...')

        ckpt = tf.train.get_checkpoint_state(self.model_out_dir)
        if ckpt and ckpt.model_checkpoint_path:
            ckpt_name = os.path.basename(ckpt.model_checkpoint_path)
            self.saver.restore(self.sess, os.path.join(self.model_out_dir,
                ckpt_name))

            self.best_auc_sum = self.sess.run(self.model.best_auc_sum)
            print('=====================================================')
            print('                      Model saved!                   ')
            print(' Best auc_sum: {:.3}'.format(self.best_auc_sum))
            print('=====================================================')

            return True
        else:
            return False
```

6.5 Model

This is the most crucial module of the project where the Neural Network is build. Our Model consists of two neural networks Discriminator and Generator both of which are defined here. The Primary task is extraction of features using CGAN and reconstruct the image using the feature Vector. The module also comprises of function which are used to train the neural network

```
1
2  import tensorflow as tf
3  # noinspection PyPep8Naming
4  import TensorFlow_utils as tf_utils
5
6
7  class CGAN(object):
8      def __init__(self, sess, flags, image_size):
9          self.sess = sess
10         self.flags = flags
11         self.image_size = image_size
12
13         self.alpha_recip = 1. / self.flags.ratio_gan2seg if self.flags.
               ratio_gan2seg > 0 else 0
14         self._gen_train_ops, self._dis_train_ops = [], []
15         self.gen_c, self.dis_c = 32, 32
16
17         self._build_net()
18         self._init_assign_op()  # initialize assign operations
19
20         print('Initialized CGAN SUCCESS!\n')
21
22     def _build_net(self):
23         self.X = tf.placeholder(tf.float32, shape=[None, *self.image_size, 3],
               name='image')
24         self.Y = tf.placeholder(tf.float32, shape=[None, *self.image_size, 1],
               name='vessel')
25
26         self.g_samples = self.generator(self.X)
27         self.real_pair = tf.concat([self.X, self.Y], axis=3)
28         self.fake_pair = tf.concat([self.X, self.g_samples], axis=3)
29
30         d_real, d_logit_real = self.discriminator(self.real_pair)
31         d_fake, d_logit_fake = self.discriminator(self.fake_pair, is_reuse=True)
32
33         # discrminator loss
34         self.d_loss_real = tf.reduce_mean(tf.nn.
               sigmoid_cross_entropy_with_logits(
35              logits=d_logit_real, labels=tf.ones_like(d_real)))
36         self.d_loss_fake = tf.reduce_mean(tf.nn.
               sigmoid_cross_entropy_with_logits(
37              logits=d_logit_fake, labels=tf.zeros_like(d_logit_fake)))
38         self.d_loss = self.d_loss_real + self.d_loss_fake
39
40         # generator loss
41         gan_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(
42              logits=d_logit_fake, labels=tf.ones_like(d_logit_fake)))
43         seg_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits
               =self.g_samples, labels=self.Y))
44         self.g_loss = self.alpha_recip * gan_loss + seg_loss
45
```

```
        t_vars = tf.trainable_variables()
        d_vars = [var for var in t_vars if 'd_' in var.name]
        g_vars = [var for var in t_vars if 'g_' in var.name]

        dis_op = tf.train.AdamOptimizer(learning_rate=self.flags.learning_rate,
            beta1=self.flags.beta1)\
             .minimize(self.d_loss, var_list=d_vars)
        dis_ops = [dis_op] + self._dis_train_ops
        self.dis_optim = tf.group(*dis_ops)

        gen_op = tf.train.AdamOptimizer(learning_rate=self.flags.learning_rate,
            beta1=self.flags.beta1)\
             .minimize(self.g_loss, var_list=g_vars)
        gen_ops = [gen_op] + self._gen_train_ops
        self.gen_optim = tf.group(*gen_ops)

    def _init_assign_op(self):
        self.best_auc_sum_placeholder = tf.placeholder(tf.float32, name='
            best_auc_sum_placeholder')
        self.auc_pr_placeholder = tf.placeholder(tf.float32, name='
            auc_pr_placeholder')
        self.auc_roc_placeholder = tf.placeholder(tf.float32, name='
            auc_roc_placeholder')
        self.dice_coeff_placeholder = tf.placeholder(tf.float32, name='
            dice_coeff_placeholder')
        self.acc_placeholder = tf.placeholder(tf.float32, name='acc_placeholder'
            )
        self.sensitivity_placeholder = tf.placeholder(tf.float32, name='
            sensitivity_placeholder')
        self.specificity_placeholder = tf.placeholder(tf.float32, name='
            specificity_placeholder')
        self.score_placeholder = tf.placeholder(tf.float32, name='
            score_placeholder')

        self.best_auc_sum = tf.Variable(0., trainable=False, dtype=tf.float32,
            name='best_auc_sum')
        auc_pr = tf.Variable(0., trainable=False, dtype=tf.float32, name='auc_pr
            ')
        auc_roc = tf.Variable(0., trainable=False, dtype=tf.float32, name='
            auc_roc')
        dice_coeff = tf.Variable(0., trainable=False, dtype=tf.float32, name='
            dice_coeff')
        acc = tf.Variable(0., trainable=False, dtype=tf.float32, name='acc')
        sensitivity = tf.Variable(0., trainable=False, dtype=tf.float32, name='
            sensitivity')
        specificity = tf.Variable(0., trainable=False, dtype=tf.float32, name='
            specificity')
        score = tf.Variable(0., trainable=False, dtype=tf.float32, name='score')

        self.best_auc_sum_assign_op = self.best_auc_sum.assign(self.
            best_auc_sum_placeholder)
        auc_pr_assign_op = auc_pr.assign(self.auc_pr_placeholder)
        auc_roc_assign_op = auc_roc.assign(self.auc_roc_placeholder)
        dice_coeff_assign_op = dice_coeff.assign(self.dice_coeff_placeholder)
        acc_assign_op = acc.assign(self.acc_placeholder)
        sensitivity_assign_op = sensitivity.assign(self.sensitivity_placeholder)
        specificity_assign_op = specificity.assign(self.specificity_placeholder)
        score_assign_op = score.assign(self.score_placeholder)

        self.measure_assign_op = tf.group(auc_pr_assign_op, auc_roc_assign_op,
            dice_coeff_assign_op,
```

```
                                        acc_assign_op, sensitivity_assign_op,
                                            specificity_assign_op,
                                        score_assign_op)

        # for tensorboard
        if not self.flags.is_test:
            self.writer = tf.summary.FileWriter("{}/logs/{}_{}_{}".format(
                self.flags.dataset, self.flags.discriminator, self.flags.
                    train_interval, self.flags.batch_size))

        auc_pr_summ = tf.summary.scalar("auc_pr_summary", auc_pr)
        auc_roc_summ = tf.summary.scalar("auc_roc_summary", auc_roc)
        dice_coeff_summ = tf.summary.scalar("dice_coeff_summary", dice_coeff)
        acc_summ = tf.summary.scalar("acc_summary", acc)
        sensitivity_summ = tf.summary.scalar("sensitivity_summary", sensitivity)
        specificity_summ = tf.summary.scalar("specificity_summary", specificity)
        score_summ = tf.summary.scalar("score_summary", score)

        self.measure_summary = tf.summary.merge([auc_pr_summ, auc_roc_summ,
            dice_coeff_summ, acc_summ,
                                                 sensitivity_summ,
                                                    specificity_summ,
                                                    score_summ])

    def generator(self, data, name='g_'):
        with tf.variable_scope(name):
            # conv1: (N, 640, 640, 1) -> (N, 320, 320, 32)
            conv1 = tf_utils.conv2d(data, self.gen_c, k_h=3, k_w=3, d_h=1, d_w
                =1, name='conv1_conv1')
            conv1 = tf_utils.batch_norm(conv1, name='conv1_batch1', _ops=self.
                _gen_train_ops)
            conv1 = tf.nn.relu(conv1, name='conv1_relu1')
            conv1 = tf_utils.conv2d(conv1, self.gen_c, k_h=3, k_w=3, d_h=1, d_w
                =1, name='conv1_conv2')
            conv1 = tf_utils.batch_norm(conv1, name='conv1_batch2', _ops=self.
                _gen_train_ops)
            conv1 = tf.nn.relu(conv1, name='conv1_relu2')
            pool1 = tf_utils.max_pool_2x2(conv1, name='maxpool1')

            # conv2: (N, 320, 320, 32) -> (N, 160, 160, 64)
            conv2 = tf_utils.conv2d(pool1, 2*self.gen_c, k_h=3, k_w=3, d_h=1,
                d_w=1, name='conv2_conv1')
            conv2 = tf_utils.batch_norm(conv2, name='conv2_batch1', _ops=self.
                _gen_train_ops)
            conv2 = tf.nn.relu(conv2, name='conv2_relu1')
            conv2 = tf_utils.conv2d(conv2, 2*self.gen_c, k_h=3, k_w=3, d_h=1,
                d_w=1, name='conv2_conv2')
            conv2 = tf_utils.batch_norm(conv2, name='conv2-batch2', _ops=self.
                _gen_train_ops)
            conv2 = tf.nn.relu(conv2, name='conv2_relu2')
            pool2 = tf_utils.max_pool_2x2(conv2, name='maxpool2')

            # conv3: (N, 160, 160, 64) -> (N, 80, 80, 128)
            conv3 = tf_utils.conv2d(pool2, 4*self.gen_c, k_h=3, k_w=3, d_h=1,
                d_w=1, name='conv3_conv1')
            conv3 = tf_utils.batch_norm(conv3, name='conv3_batch1', _ops=self.
                _gen_train_ops)
            conv3 = tf.nn.relu(conv3, name='conv3_relu1')
            conv3 = tf_utils.conv2d(conv3, 4*self.gen_c, k_h=3, k_w=3, d_h=1,
                d_w=1, name='conv3_conv2')
            conv3 = tf_utils.batch_norm(conv3, name='conv3_batch2', _ops=self.
```

```
        _gen_train_ops)
    conv3 = tf.nn.relu(conv3, name='conv3_relu2')
    pool3 = tf_utils.max_pool_2x2(conv3, name='maxpool3')

    # conv4: (N, 80, 80, 128) -> (N, 40, 40, 256)
    conv4 = tf_utils.conv2d(pool3, 8*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv4_conv1')
    conv4 = tf_utils.batch_norm(conv4, name='conv4_batch1', _ops=self.
        _gen_train_ops)
    conv4 = tf.nn.relu(conv4, name='conv4_relu1')
    conv4 = tf_utils.conv2d(conv4, 8*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv4_conv2')
    conv4 = tf_utils.batch_norm(conv4, name='conv4_batch2', _ops=self.
        _gen_train_ops)
    conv4 = tf.nn.relu(conv4, name='conv4_relu2')
    pool4 = tf_utils.max_pool_2x2(conv4, name='maxpool4')

    # conv5: (N, 40, 40, 256) -> (N, 40, 40, 512)
    conv5 = tf_utils.conv2d(pool4, 16*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv5_conv1')
    conv5 = tf_utils.batch_norm(conv5, name='conv5_batch1', _ops=self.
        _gen_train_ops)
    conv5 = tf.nn.relu(conv5, name='conv5_relu1')
    conv5 = tf_utils.conv2d(conv5, 16*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv5_conv2')
    conv5 = tf_utils.batch_norm(conv5, name='conv5_batch2', _ops=self.
        _gen_train_ops)
    conv5 = tf.nn.relu(conv5, name='conv5_relu2')

    # conv6: (N, 40, 40, 512) -> (N, 80, 80, 256)
    up1 = tf_utils.upsampling2d(conv5, size=(2, 2), name='conv6_up')
    conv6 = tf.concat([up1, conv4], axis=3, name='conv6_concat')
    conv6 = tf_utils.conv2d(conv6, 8*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv6_conv1')
    conv6 = tf_utils.batch_norm(conv6, name='conv6_batch1', _ops=self.
        _gen_train_ops)
    conv6 = tf.nn.relu(conv6, name='conv6_relu1')
    conv6 = tf_utils.conv2d(conv6, 8*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv6_conv2')
    conv6 = tf_utils.batch_norm(conv6, name='conv6_batch2', _ops=self.
        _gen_train_ops)
    conv6 = tf.nn.relu(conv6, name='conv6_relu2')

    # conv7: (N, 80, 80, 256) -> (N, 160, 160, 128)
    up2 = tf_utils.upsampling2d(conv6, size=(2, 2), name='conv7_up')
    conv7 = tf.concat([up2, conv3], axis=3, name='conv7_concat')
    conv7 = tf_utils.conv2d(conv7, 4*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv7_conv1')
    conv7 = tf_utils.batch_norm(conv7, name='conv7_batch1', _ops=self.
        _gen_train_ops)
    conv7 = tf.nn.relu(conv7, name='conv7_relu1')
    conv7 = tf_utils.conv2d(conv7, 4*self.gen_c, k_h=3, k_w=3, d_h=1,
        d_w=1, name='conv7_conv2')
    conv7 = tf_utils.batch_norm(conv7, name='conv7_batch2', _ops=self.
        _gen_train_ops)
    conv7 = tf.nn.relu(conv7, name='conv7_relu2')

    # conv8: (N, 160, 160, 128) -> (N, 320, 320, 64)
    up3 = tf_utils.upsampling2d(conv7, size=(2, 2), name='conv8_up')
    conv8 = tf.concat([up3, conv2], axis=3, name='conv8_concat')
    conv8 = tf_utils.conv2d(conv8, 2*self.gen_c, k_h=3, k_w=3, d_h=1,
```

```
                d_w=1, name='conv8_conv1')
178             conv8 = tf_utils.batch_norm(conv8, name='conv8_batch1', _ops=self.
                    _gen_train_ops)
179             conv8 = tf.nn.relu(conv8, name='conv8_relu1')
180             conv8 = tf_utils.conv2d(conv8, 2*self.gen_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv8_conv2')
181             conv8 = tf_utils.batch_norm(conv8, name='conv8_batch2', _ops=self.
                    _gen_train_ops)
182             conv8 = tf.nn.relu(conv8, name='conv8_relu2')
183
184             # conv9: (N, 320, 320, 64) -> (N, 640, 640, 32)
185             up4 = tf_utils.upsampling2d(conv8, size=(2, 2), name='conv9_up')
186             conv9 = tf.concat([up4, conv1], axis=3, name='conv9_concat')
187             conv9 = tf_utils.conv2d(conv9, self.gen_c, k_h=3, k_w=3, d_h=1, d_w
                    =1, name='conv9_conv1')
188             conv9 = tf_utils.batch_norm(conv9, name='conv9_batch1', _ops=self.
                    _gen_train_ops)
189             conv9 = tf.nn.relu(conv9, name='conv9_relu1')
190             conv9 = tf_utils.conv2d(conv9, self.gen_c, k_h=3, k_w=3, d_h=1, d_w
                    =1, name='conv9_conv2')
191             conv9 = tf_utils.batch_norm(conv9, name='conv9_batch2', _ops=self.
                    _gen_train_ops)
192             conv9 = tf.nn.relu(conv9, name='conv9_relu2')
193
194             # output layer: (N, 640, 640, 32) -> (N, 640, 640, 1)
195             output = tf_utils.conv2d(conv9, 1, k_h=1, k_w=1, d_h=1, d_w=1, name=
                    'conv_output')
196
197             return tf.nn.sigmoid(output)
198
199     def discriminator(self, data, is_reuse=False):
200         if self.flags.discriminator == 'pixel':
201             return self.discriminator_pixel(data, is_reuse=is_reuse)
202         elif self.flags.discriminator == 'patch1':
203             return self.discriminator_patch1(data, is_reuse=is_reuse)
204         elif self.flags.discriminator == 'patch2':
205             return self.discriminator_patch2(data, is_reuse=is_reuse)
206         elif self.flags.discriminator == 'image':
207             return self.discriminator_image(data, is_reuse=is_reuse)
208         else:
209             raise NotImplementedError
210
211     def discriminator_pixel(self, data, name='d_', is_reuse=False):
212         with tf.variable_scope(name) as scope:
213             if is_reuse is True:
214                 scope.reuse_variables()
215
216             # conv1: (N, 640, 640, 4) -> (N,, 640, 640, 32)
217             conv1 = tf_utils.conv2d(data, self.dis_c, k_h=3, k_w=3, d_h=1, d_w
                    =1, name='conv1_conv1')
218             conv1 = tf_utils.lrelu(conv1, name='conv1_lrelu1')
219
220             # conv2: (N, 640, 640, 32) -> (N, 640, 640, 64)
221             conv2 = tf_utils.conv2d(conv1, 2*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv2_conv1')
222             conv2 = tf_utils.lrelu(conv2)
223
224             # conv3: (N, 640, 640, 64) -> (N, 640, 640, 128)
225             conv3 = tf_utils.conv2d(conv2, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv1')
226             conv3 = tf_utils.lrelu(conv3)
```

```
227
228             # output layer: (N, 640, 640, 128) -> (N, 640, 640, 1)
229             output = tf_utils.conv2d(conv3, 1, k_h=1, k_w=1, d_h=1, d_w=1, name=
                    'conv_output')
230
231             return tf.nn.sigmoid(output), output
232
233     def discriminator_patch2(self, data, name='d_', is_reuse=False):
234         with tf.variable_scope(name) as scope:
235             if is_reuse is True:
236                 scope.reuse_variables()
237
238             # conv1: (N, 640, 640, 4) -> (N,, 160, 160, 32)
239             conv1 = tf_utils.conv2d(data, self.dis_c, k_h=3, k_w=3, d_h=2, d_w
                    =2, name='conv1_conv1')
240             conv1 = tf_utils.batch_norm(conv1, name='conv1_batch1', _ops=self.
                    _dis_train_ops)
241             conv1 = tf.nn.relu(conv1, name='conv1_relu1')
242             conv1 = tf_utils.conv2d(conv1, self.dis_c, k_h=3, k_w=3, d_h=1, d_w
                    =1, name='conv1_conv2')
243             conv1 = tf_utils.batch_norm(conv1, name='conv1_batch2', _ops=self.
                    _dis_train_ops)
244             conv1 = tf.nn.relu(conv1, name='conv1_relu2')
245             pool1 = tf_utils.max_pool_2x2(conv1, name='maxpool1')
246
247             # conv2: (N, 160, 160, 32) -> (N, 80, 80, 64)
248             conv2 = tf_utils.conv2d(pool1, 2*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv2_conv1')
249             conv2 = tf_utils.batch_norm(conv2, name='conv2_batch1', _ops=self.
                    _dis_train_ops)
250             conv2 = tf.nn.relu(conv2, name='conv2_relu1')
251             conv2 = tf_utils.conv2d(conv2, 2*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv2_conv2')
252             conv2 = tf_utils.batch_norm(conv2, name='conv2_batch2', _ops=self.
                    _dis_train_ops)
253             conv2 = tf.nn.relu(conv2, name='conv2_relu2')
254             pool2 = tf_utils.max_pool_2x2(conv2, name='maxpool2')
255
256             # conv3: (N, 80, 80, 64) -> (N, 80, 80, 128)
257             conv3 = tf_utils.conv2d(pool2, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv1')
258             conv3 = tf_utils.batch_norm(conv3, name='conv3_batch1', _ops=self.
                    _dis_train_ops)
259             conv3 = tf.nn.relu(conv3, name='conv3_relu1')
260             conv3 = tf_utils.conv2d(conv3, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv2')
261             conv3 = tf_utils.batch_norm(conv3, name='conv3_batch2', _ops=self.
                    _dis_train_ops)
262             conv3 = tf.nn.relu(conv3, name='conv3_relu2')
263
264             # output layer: (N, 80, 80, 128) -> (N, 80, 80, 1)
265             output = tf_utils.conv2d(conv3, 1, k_h=1, k_w=1, d_h=1, d_w=1, name=
                    'conv_output')
266
267             return tf.nn.sigmoid(output), output
268
269     def discriminator_patch1(self, data, name='d_', is_reuse=False):
270         with tf.variable_scope(name) as scope:
271             if is_reuse is True:
272                 scope.reuse_variables()
273
```

```
274             # conv1: (N, 640, 640, 4) -> (N,, 160, 160, 32)
275             conv1 = tf_utils.conv2d(data, self.dis_c, k_h=3, k_w=3, d_h=2, d_w
                    =2, name='conv1_conv1')
276             conv1 = tf_utils.batch_norm(conv1, name='conv1_batch1', _ops=self.
                    _dis_train_ops)
277             conv1 = tf.nn.relu(conv1, name='conv1_relu1')
278             conv1 = tf_utils.conv2d(conv1, self.dis_c, k_h=3, k_w=3, d_h=1, d_w
                    =1, name='conv1_conv2')
279             conv1 = tf_utils.batch_norm(conv1, name='conv1_batch2', _ops=self.
                    _dis_train_ops)
280             conv1 = tf.nn.relu(conv1, name='conv1_relu2')
281             pool1 = tf_utils.max_pool_2x2(conv1, name='maxpool1')
282
283             # conv2: (N, 160, 160, 32) -> (N, 40, 40, 64)
284             conv2 = tf_utils.conv2d(pool1, 2*self.dis_c, k_h=3, k_w=3, d_h=2,
                    d_w=2, name='conv2_conv1')
285             conv2 = tf_utils.batch_norm(conv2, name='conv2_batch1', _ops=self.
                    _dis_train_ops)
286             conv2 = tf.nn.relu(conv2, name='conv2_relu1')
287             conv2 = tf_utils.conv2d(conv2, 2*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv2_conv2')
288             conv2 = tf_utils.batch_norm(conv2, name='conv2_batch2', _ops=self.
                    _dis_train_ops)
289             conv2 = tf.nn.relu(conv2, name='conv2_relu2')
290             pool2 = tf_utils.max_pool_2x2(conv2, name='maxpool2')
291
292             # conv3: (N, 40, 40, 64) -> (N, 20, 20, 128)
293             conv3 = tf_utils.conv2d(pool2, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv1')
294             conv3 = tf_utils.batch_norm(conv3, name='conv3_batch1', _ops=self.
                    _dis_train_ops)
295             conv3 = tf.nn.relu(conv3, name='conv3_relu1')
296             conv3 = tf_utils.conv2d(conv3, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv2')
297             conv3 = tf_utils.batch_norm(conv3, name='conv3_batch2', _ops=self.
                    _dis_train_ops)
298             conv3 = tf.nn.relu(conv3, name='conv3_relu2')
299             pool3 = tf_utils.max_pool_2x2(conv3, name='maxpool3')
300
301             # conv4: (N, 20, 20, 128) -> (N, 10, 10, 256)
302             conv4 = tf_utils.conv2d(pool3, 8*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv4_conv1')
303             conv4 = tf_utils.batch_norm(conv4, name='conv4_batch1', _ops=self.
                    _dis_train_ops)
304             conv4 = tf.nn.relu(conv4, name='conv4_relu1')
305             conv4 = tf_utils.conv2d(conv4, 8*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv4_conv2')
306             conv4 = tf_utils.batch_norm(conv4, name='conv4_batch2', _ops=self.
                    _dis_train_ops)
307             conv4 = tf.nn.relu(conv4, name='conv4_relu2')
308             pool4 = tf_utils.max_pool_2x2(conv4, name='maxpool4')
309
310             # conv5: (N, 10, 10, 256) -> (N, 10, 10, 512)
311             conv5 = tf_utils.conv2d(pool4, 16*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv5_conv1')
312             conv5 = tf_utils.batch_norm(conv5, name='conv5_batch1', _ops=self.
                    _dis_train_ops)
313             conv5 = tf.nn.relu(conv5, name='conv5_relu1')
314             conv5 = tf_utils.conv2d(conv5, 16*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv5_conv2')
315             conv5 = tf_utils.batch_norm(conv5, name='conv5_batch2', _ops=self.
```

```
                _dis_train_ops)
316             conv5 = tf.nn.relu(conv5, name='conv5_relu2')
317
318             # output layer: (N, 10, 10, 512) -> (N, 10, 10, 1)
319             output = tf_utils.conv2d(conv5, 1, k_h=1, k_w=1, d_h=1, d_w=1, name=
                    'conv_output')
320
321             return tf.nn.sigmoid(output), output
322
323     def discriminator_image(self, data, name='d_', is_reuse=False):
324         with tf.variable_scope(name) as scope:
325             if is_reuse is True:
326                 scope.reuse_variables()
327
328             # conv1: (N, 640, 640, 4) -> (N,, 160, 160, 32)
329             conv1 = tf_utils.conv2d(data, self.dis_c, k_h=3, k_w=3, d_h=2, d_w
                    =2, name='conv1_conv1')
330             conv1 = tf_utils.batch_norm(conv1, name='conv1_batch1', _ops=self.
                    _dis_train_ops)
331             conv1 = tf.nn.relu(conv1, name='conv1_relu1')
332             conv1 = tf_utils.conv2d(conv1, self.dis_c, k_h=3, k_w=3, d_h=1, d_w
                    =1, name='conv1_conv2')
333             conv1 = tf_utils.batch_norm(conv1, name='conv1_batch2', _ops=self.
                    _dis_train_ops)
334             conv1 = tf.nn.relu(conv1, name='conv1_relu2')
335             pool1 = tf_utils.max_pool_2x2(conv1, name='maxpool1')
336
337             # conv2: (N, 160, 160, 32) -> (N, 40, 40, 64)
338             conv2 = tf_utils.conv2d(pool1, 2*self.dis_c, k_h=3, k_w=3, d_h=2,
                    d_w=2, name='conv2_conv1')
339             conv2 = tf_utils.batch_norm(conv2, name='conv2_batch1', _ops=self.
                    _dis_train_ops)
340             conv2 = tf.nn.relu(conv2, name='conv2_relu1')
341             conv2 = tf_utils.conv2d(conv2, 2*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv2_conv2')
342             conv2 = tf_utils.batch_norm(conv2, name='conv2_batch2', _ops=self.
                    _dis_train_ops)
343             conv2 = tf.nn.relu(conv2, name='conv2_relu2')
344             pool2 = tf_utils.max_pool_2x2(conv2, name='maxpool2')
345
346             # conv3: (N, 40, 40, 64) -> (N, 20, 20, 128)
347             conv3 = tf_utils.conv2d(pool2, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv1')
348             conv3 = tf_utils.batch_norm(conv3, name='conv3_batch1', _ops=self.
                    _dis_train_ops)
349             conv3 = tf.nn.relu(conv3, name='conv3_relu1')
350             conv3 = tf_utils.conv2d(conv3, 4*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv3_conv2')
351             conv3 = tf_utils.batch_norm(conv3, name='conv3_batch2', _ops=self.
                    _dis_train_ops)
352             conv3 = tf.nn.relu(conv3, name='conv3_relu2')
353             pool3 = tf_utils.max_pool_2x2(conv3, name='maxpool3')
354
355             # conv4: (N, 20, 20, 128) -> (N, 10, 10, 256)
356             conv4 = tf_utils.conv2d(pool3, 8*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv4_conv1')
357             conv4 = tf_utils.batch_norm(conv4, name='conv4_batch1', _ops=self.
                    _dis_train_ops)
358             conv4 = tf.nn.relu(conv4, name='conv4_relu1')
359             conv4 = tf_utils.conv2d(conv4, 8*self.dis_c, k_h=3, k_w=3, d_h=1,
                    d_w=1, name='conv4_conv2')
```

```
            conv4 = tf_utils.batch_norm(conv4, name='conv4_batch2', _ops=self._dis_train_ops)
            conv4 = tf.nn.relu(conv4, name='conv4_relu2')
            pool4 = tf_utils.max_pool_2x2(conv4, name='maxpool4')

            # conv5: (N, 10, 10, 256) -> (N, 10, 10, 512)
            conv5 = tf_utils.conv2d(pool4, 16*self.dis_c, k_h=3, k_w=3, d_h=1, d_w=1, name='conv5_conv1')
            conv5 = tf_utils.batch_norm(conv5, name='conv5_batch1', _ops=self._dis_train_ops)
            conv5 = tf.nn.relu(conv5, name='conv5_relu1')
            conv5 = tf_utils.conv2d(conv5, 16*self.dis_c, k_h=3, k_w=3, d_h=1, d_w=1, name='conv5_conv2')
            conv5 = tf_utils.batch_norm(conv5, name='conv5_batch2', _ops=self._dis_train_ops)
            conv5 = tf.nn.relu(conv5, name='conv5_relu2')

            # output layer: (N, 10, 10, 512) -> (N, 1, 1, 512) -> (N, 1)
            shape = conv5.get_shape().as_list()
            gap = tf.layers.average_pooling2d(inputs=conv5, pool_size=shape[1], strides=1, padding='VALID',
                                              name='global_vaerage_pool')
            gap_flatten = tf.reshape(gap, [-1, 16*self.dis_c])
            output = tf_utils.linear(gap_flatten, 1, name='linear_output')

            return tf.nn.sigmoid(output), output

    def train_dis(self, x_data, y_data):
        feed_dict = {self.X: x_data, self.Y: y_data}
        # run discriminator
        _, d_loss = self.sess.run([self.dis_optim, self.d_loss], feed_dict=feed_dict)

        return d_loss

    def train_gen(self, x_data, y_data):
        feed_dict = {self.X: x_data, self.Y: y_data}
        # run generator
        _, g_loss = self.sess.run([self.gen_optim, self.g_loss], feed_dict=feed_dict)

        return g_loss

    def measure_assign(self, auc_pr, auc_roc, dice_coeff, acc, sensitivity, specificity, score, iter_time):
        feed_dict = {self.auc_pr_placeholder: auc_pr,
                     self.auc_roc_placeholder: auc_roc,
                     self.dice_coeff_placeholder: dice_coeff,
                     self.acc_placeholder: acc,
                     self.sensitivity_placeholder: sensitivity,
                     self.specificity_placeholder: specificity,
                     self.score_placeholder: score}

        self.sess.run(self.measure_assign_op, feed_dict=feed_dict)

        summary = self.sess.run(self.measure_summary)
        self.writer.add_summary(summary, iter_time)

    def best_auc_sum_assign(self, auc_sum):
        self.sess.run(self.best_auc_sum_assign_op, feed_dict={self.best_auc_sum_placeholder: auc_sum})
```

```
411
412     def sample_imgs(self, x_data):
413         return self.sess.run(self.g_samples, feed_dict={self.X: x_data})
```

6.6 Tensorflow Utils

This module has rewritten library function for ease of use according to the requirements

```
import tensorflow as tf
import tensorflow.contrib.slim as slim
from tensorflow.python.training import moving_averages


def linear(input_, output_size, stddev=0.02, bias_start=0.0, with_w=False, name='fc'):
    shape = input_.get_shape().as_list()
    # print('shape: ', shape)

    with tf.variable_scope(name) as scope:
        matrix = tf.get_variable(name="matrix", shape=[shape[1], output_size],
                                 dtype=tf.float32, initializer=tf.contrib.layers.xavier_initializer())
        bias = tf.get_variable(name="bias", shape=[output_size],
                               initializer=tf.constant_initializer(bias_start))
        if with_w:
            return tf.matmul(input_, matrix) + bias, matrix, bias
        else:
            return tf.matmul(input_, matrix) + bias


def batch_norm(x, name, _ops, is_train=True):
    """Batch normalization."""
    with tf.variable_scope(name):
        params_shape = [x.get_shape()[-1]]

        beta = tf.get_variable('beta', params_shape, tf.float32,
                               initializer=tf.constant_initializer(0.0, tf.float32))
        gamma = tf.get_variable('gamma', params_shape, tf.float32,
                                initializer=tf.constant_initializer(1.0, tf.float32))

        if is_train is True:
            mean, variance = tf.nn.moments(x, [0, 1, 2], name='moments')

            moving_mean = tf.get_variable('moving_mean', params_shape, tf.float32,
                                          initializer=tf.constant_initializer(0.0, tf.float32),
                                          trainable=False)
            moving_variance = tf.get_variable('moving_variance', params_shape, tf.float32,
                                              initializer=tf.constant_initializer(1.0, tf.float32),
                                              trainable=False)

            _ops.append(moving_averages.assign_moving_average(moving_mean, mean, 0.9))
            _ops.append(moving_averages.assign_moving_average(moving_variance, variance, 0.9))
        else:
            mean = tf.get_variable('moving_mean', params_shape, tf.float32,
```

```
                                    initializer=tf.constant_initializer(0.0, tf.
                                        float32), trainable=False)
            variance = tf.get_variable('moving_variance', params_shape, tf.
                float32, trainable=False)

        # epsilon used to be 1e-5. Maybe 0.001 solves NaN problem in deeper net.
        y = tf.nn.batch_normalization(x, mean, variance, beta, gamma, 1e-5)
        y.set_shape(x.get_shape())

        return y


def conv2d(input_, output_dim, k_h=5, k_w=5, d_h=2, d_w=2, stddev=0.02, name='
    conv2d'):
    with tf.variable_scope(name):
        w = tf.get_variable('w', [k_h, k_w, input_.get_shape()[-1], output_dim],
                            initializer=tf.truncated_normal_initializer(stddev=
                                stddev))
        conv = tf.nn.conv2d(input_, w, strides=[1, d_h, d_w, 1], padding='SAME')

        biases = tf.get_variable('biases', [output_dim], initializer=tf.
            constant_initializer(0.0))
        # conv = tf.reshape(tf.nn.bias_add(conv, biases), conv.get_shape())
        conv = tf.nn.bias_add(conv, biases)

        return conv


def deconv2d(input_, output_shape, k_h=5, k_w=5, d_h=2, d_w=2, stddev=0.02, name
    ='deconv2d', with_w=False):
    with tf.variable_scope(name):
        # filter : [height, width, output_channels, in_channels]
        w = tf.get_variable('w', [k_h, k_w, output_shape[-1], input_.get_shape()
            [-1]],
                            initializer=tf.random_normal_initializer(stddev=
                                stddev))
        deconv = tf.nn.conv2d_transpose(input_, w, output_shape=output_shape,
            strides=[1, d_h, d_w, 1])

        biases = tf.get_variable('biases', [output_shape[-1]], initializer=tf.
            constant_initializer(0.0))

        # deconv = tf.reshape(tf.nn.bias_add(deconv, biases), deconv.get_shape()
            )
        deconv = tf.nn.bias_add(deconv, biases)

        if with_w:
            return deconv, w, biases
        else:
            return deconv


def upsampling2d(input_, size=(2, 2), name='upsampling2d'):
    with tf.name_scope(name):
        shape = input_.get_shape().as_list()
        return tf.image.resize_nearest_neighbor(input_, size=(size[0] * shape
            [1], size[1] * shape[2]))


def max_pool_2x2(x, name='max_pool'):
    with tf.name_scope(name):
```

```
95          return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
                padding='SAME')
96
97
98  def lrelu(x, leak=0.2, name='lrelu'):
99      return tf.maximum(x, leak*x, name=name)
100
101
102 def xavier_init(in_dim):
103     print('in_dim: ', in_dim)
104     xavier_stddev = 1. / tf.sqrt(in_dim / 2.)
105     return xavier_stddev
106
107
108 def print_activations(t):
109     print(t.op.name, ' ', t.get_shape().as_list())
110
111
112 def show_all_variables():
113     model_vars = tf.trainable_variables()
114     slim.model_analyzer.analyze_vars(model_vars, print_info=True)
```

6.7 utils

The module is toolbox for other modules as all the accessory functions are written here such as performing operations on the dataset, fetching images from the dataset, Calculation of various accuracy of parameters are done here such as Sensitivity, Specificity, f1 score , Precision Recall, dice coefficient.

```
1
2  import os
3  import sys
4
5  import pickle
6  import matplotlib
7  import numpy as np
8  import matplotlib.pyplot as plt
9
10 from PIL import Image, ImageEnhance
11 from skimage import filters
12 from scipy.ndimage import rotate
13 from sklearn.metrics import roc_curve, roc_auc_score, precision_recall_curve,
       auc, confusion_matrix
14
15
16 def get_img_path(target_dir, dataset):
17     img_files, vessel_files, mask_files = None, None, None
18     if dataset == 'DRIVE':
19         img_files, vessel_files, mask_files = DRIVE_files(target_dir)
20     elif dataset == 'STARE':
21         img_files, vessel_files, mask_files = STARE_files(target_dir)
22
23     return img_files, vessel_files, mask_files
24
25
26 # noinspection PyPep8Naming
27 def STARE_files(data_path):
28     img_dir = os.path.join(data_path, "images")
29     vessel_dir = os.path.join(data_path, "1st_manual")
30     mask_dir = os.path.join(data_path, "mask")
31
32     img_files = all_files_under(img_dir, extension=".ppm")
33     vessel_files = all_files_under(vessel_dir, extension=".ppm")
34     mask_files = all_files_under(mask_dir, extension=".ppm")
35
36     return img_files, vessel_files, mask_files
37
38
39 # noinspection PyPep8Naming
40 def DRIVE_files(data_path):
41     img_dir = os.path.join(data_path, "images")
42     vessel_dir = os.path.join(data_path, "1st_manual")
43     mask_dir = os.path.join(data_path, "mask")
44
45     img_files = all_files_under(img_dir, extension=".tif")
46     vessel_files = all_files_under(vessel_dir, extension=".gif")
47     mask_files = all_files_under(mask_dir, extension=".gif")
48
49     return img_files, vessel_files, mask_files
50
51
52 def load_images_under_dir(path_dir):
```

```
    files = all_files_under(path_dir)
    return imagefiles2arrs(files)


def all_files_under(path, extension=None, append_path=True, sort=True):
    if append_path:
        if extension is None:
            filenames = [os.path.join(path, fname) for fname in os.listdir(path)
                ]
        else:
            filenames = [os.path.join(path, fname)
                         for fname in os.listdir(path) if fname.endswith(
                             extension)]
    else:
        if extension is None:
            filenames = [os.path.basename(fname) for fname in os.listdir(path)]
        else:
            filenames = [os.path.basename(fname)
                         for fname in os.listdir(path) if fname.endswith(
                             extension)]

    if sort:
        filenames = sorted(filenames)

    return filenames


def imagefiles2arrs(filenames):
    img_shape = image_shape(filenames[0])
    images_arr = None

    if len(img_shape) == 3:
        images_arr = np.zeros((len(filenames), img_shape[0], img_shape[1],
            img_shape[2]), dtype=np.float32)
    elif len(img_shape) == 2:
        images_arr = np.zeros((len(filenames), img_shape[0], img_shape[1]),
            dtype=np.float32)

    for file_index in range(len(filenames)):
        img = Image.open(filenames[file_index])
        images_arr[file_index] = np.asarray(img).astype(np.float32)

    return images_arr


def get_train_batch(train_img_files, train_vessel_files, train_indices, img_size
    ):
    batch_size = len(train_indices)
    batch_img_files, batch_vessel_files = [], []
    for _, idx in enumerate(train_indices):
        batch_img_files.append(train_img_files[idx])
        batch_vessel_files.append(train_vessel_files[idx])

    # load images
    fundus_imgs = imagefiles2arrs(batch_img_files)
    vessel_imgs = imagefiles2arrs(batch_vessel_files) / 255
    fundus_imgs = pad_imgs(fundus_imgs, img_size)
    vessel_imgs = pad_imgs(vessel_imgs, img_size)
    assert (np.min(vessel_imgs) == 0 and np.max(vessel_imgs) == 1)

    # random mirror flipping
```

```
    for idx in range(batch_size):
        if np.random.random() > 0.5:
            fundus_imgs[idx] = fundus_imgs[idx, :, ::-1, :]  # flipped imgs
            vessel_imgs[idx] = vessel_imgs[idx, :, ::-1]  # flipped vessel

    # flip_index = np.random.choice(batch_size, int(np.ceil(0.5 * batch_size)),
        replace=False)
    # fundus_imgs[flip_index] = fundus_imgs[flip_index, :, ::-1, :]  # flipped
        imgs
    # vessel_imgs[flip_index] = vessel_imgs[flip_index, :, ::-1]  # flipped
        vessel

    # random rotation
    for idx in range(batch_size):
        angle = np.random.randint(360)
        fundus_imgs[idx] = random_perturbation(rotate(input=fundus_imgs[idx],
            angle=angle, axes=(0, 1),
                                                      reshape=False, order=1))
        vessel_imgs[idx] = rotate(input=vessel_imgs[idx], angle=angle, axes=(0,
            1), reshape=False, order=1)

    # z score with mean, std of each image
    for idx in range(batch_size):
        mean = np.mean(fundus_imgs[idx, ...][fundus_imgs[idx, ..., 0] > 40.0],
            axis=0)
        std = np.std(fundus_imgs[idx, ...][fundus_imgs[idx, ..., 0] > 40.0],
            axis=0)

        assert len(mean) == 3 and len(std) == 3
        fundus_imgs[idx, ...] = (fundus_imgs[idx, ...] - mean) / std

    return fundus_imgs, np.round(vessel_imgs)


def get_val_imgs(img_files, vessel_files, mask_files, img_size):
    # load images
    fundus_imgs = imagefiles2arrs(img_files)
    vessel_imgs = imagefiles2arrs(vessel_files) / 255
    mask_imgs = imagefiles2arrs(mask_files) / 255

    # padding
    fundus_imgs = pad_imgs(fundus_imgs, img_size)
    vessel_imgs = pad_imgs(vessel_imgs, img_size)
    mask_imgs = pad_imgs(mask_imgs, img_size)

    assert (np.min(vessel_imgs) == 0 and np.max(vessel_imgs) == 1)
    assert (np.min(mask_imgs) == 0 and np.max(mask_imgs) == 1)

    # augmentation
    # augment the original image (flip, rotate)
    all_fundus_imgs = [fundus_imgs]
    all_vessel_imgs = [vessel_imgs]
    all_mask_imgs = [mask_imgs]

    flipped_imgs = fundus_imgs[:, :, ::-1, :]  # flipped imgs
    flipped_vessels = vessel_imgs[:, :, ::-1]
    flipped_masks = mask_imgs[:, :, ::-1]

    all_fundus_imgs.append(flipped_imgs)
    all_vessel_imgs.append(flipped_vessels)
    all_mask_imgs.append(flipped_masks)
```

```
162
163     for angle in range(3, 360, 3):  # rotated imgs (3, 360, 3)
164         print("Val data augmentation {} degree...".format(angle))
165         all_fundus_imgs.append(random_perturbation(rotate(fundus_imgs, angle,
                axes=(1, 2), reshape=False,
166                                                           order=1)))
167         all_fundus_imgs.append(random_perturbation(rotate(flipped_imgs, angle,
                axes=(1, 2), reshape=False,
168                                                           order=1)))
169
170         all_vessel_imgs.append(rotate(vessel_imgs, angle, axes=(1, 2), reshape=
                False, order=1))
171         all_vessel_imgs.append(rotate(flipped_vessels, angle, axes=(1, 2),
                reshape=False, order=1))
172
173         all_mask_imgs.append(rotate(mask_imgs, angle, axes=(1, 2), reshape=False
                , order=1))
174         all_mask_imgs.append(rotate(flipped_masks, angle, axes=(1, 2), reshape=
                False, order=1))
175
176     fundus_imgs = np.concatenate(all_fundus_imgs, axis=0)
177     vessel_imgs = np.concatenate(all_vessel_imgs, axis=0)
178     mask_imgs = np.concatenate(all_mask_imgs, axis=0)
179
180     # z score with mean, std of each image
181     mean_std = []
182     n_all_imgs = fundus_imgs.shape[0]
183     for index in range(n_all_imgs):
184         mean = np.mean(fundus_imgs[index, ...][fundus_imgs[index, ..., 0] >
                40.0], axis=0)
185         std = np.std(fundus_imgs[index, ...][fundus_imgs[index, ..., 0] > 40.0],
                 axis=0)
186
187         assert len(mean) == 3 and len(std) == 3
188         fundus_imgs[index, ...] = (fundus_imgs[index, ...] - mean) / std
189
190         mean_std.append({'mean': mean, 'std': std})
191
192     return fundus_imgs, np.round(vessel_imgs), np.round(mask_imgs), mean_std
193
194
195 def get_test_imgs(target_dir, img_size, dataset):
196     img_files, vessel_files, mask_files, mask_imgs = None, None, None, None
197     if dataset == 'DRIVE':
198         img_files, vessel_files, mask_files = DRIVE_files(target_dir)
199     elif dataset == 'STARE':
200         img_files, vessel_files, mask_files = STARE_files(target_dir)
201
202     # load images
203     fundus_imgs = imagefiles2arrs(img_files)
204     vessel_imgs = imagefiles2arrs(vessel_files) / 255
205     fundus_imgs = pad_imgs(fundus_imgs, img_size)
206     vessel_imgs = pad_imgs(vessel_imgs, img_size)
207     assert (np.min(vessel_imgs) == 0 and np.max(vessel_imgs) == 1)
208
209     mask_imgs = imagefiles2arrs(mask_files) / 255
210     mask_imgs = pad_imgs(mask_imgs, img_size)
211     assert (np.min(mask_imgs) == 0 and np.max(mask_imgs) == 1)
212
213     # z score with mean, std of each image
214     mean_std = []
```

```
    n_all_imgs = fundus_imgs.shape[0]
    for index in range(n_all_imgs):
        mean = np.mean(fundus_imgs[index, ...][fundus_imgs[index, ..., 0] >
            40.0], axis=0)
        std = np.std(fundus_imgs[index, ...][fundus_imgs[index, ..., 0] > 40.0],
             axis=0)

        assert len(mean) == 3 and len(std) == 3
        fundus_imgs[index, ...] = (fundus_imgs[index, ...] - mean) / std

        mean_std.append({'mean': mean, 'std': std})

    return fundus_imgs, np.round(vessel_imgs), mask_imgs, mean_std


def image_shape(filename):
    img = Image.open(filename)
    img_arr = np.asarray(img)
    img_shape = img_arr.shape
    return img_shape


def pad_imgs(imgs, img_size):
    padded = None
    img_h, img_w = imgs.shape[1], imgs.shape[2]
    target_h, target_w = img_size[0], img_size[1]
    if len(imgs.shape) == 4:
        d = imgs.shape[3]
        padded = np.zeros((imgs.shape[0], target_h, target_w, d))
    elif len(imgs.shape) == 3:
        padded = np.zeros((imgs.shape[0], img_size[0], img_size[1]))

    start_h, start_w = (target_h - img_h) // 2, (target_w - img_w) // 2
    end_h, end_w = start_h + img_h, start_w + img_w
    padded[:, start_h:end_h, start_w:end_w, ...] = imgs

    return padded


def crop_to_original(imgs, ori_shape):
    # imgs: (N, 640, 640, 3 or None)
    # ori_shape: (584, 565)
    pred_shape = imgs.shape
    assert len(pred_shape) > 2

    if ori_shape == pred_shape:
        return imgs
    else:
        if len(imgs.shape) > 3:  # images (N, 640, 640, 3)
            ori_h, ori_w = ori_shape[0], ori_shape[1]
            pred_h, pred_w = pred_shape[1], pred_shape[2]

            start_h, start_w = (pred_h - ori_h) // 2, (pred_w - ori_w) // 2
            end_h, end_w = start_h + ori_h, start_w + ori_w

            return imgs[:, start_h:end_h, start_w:end_w, :]
        else:  # vesels
            ori_h, ori_w = ori_shape[0], ori_shape[1]
            pred_h, pred_w = pred_shape[1], pred_shape[2]

            start_h, start_w = (pred_h - ori_h) // 2, (pred_w - ori_w) // 2
```

```
            end_h, end_w = start_h + ori_h, start_w + ori_w

            return imgs[:, start_h:end_h, start_w:end_w]


def random_perturbation(imgs):
    for i in range(imgs.shape[0]):
        im = Image.fromarray(imgs[i, ...].astype(np.uint8))
        en = ImageEnhance.Color(im)
        im = en.enhance(np.random.uniform(0.8, 1.2))
        imgs[i, ...] = np.asarray(im).astype(np.float32)

    return imgs


def pixel_values_in_mask(true_vessels, pred_vessels, masks, split_by_img=False):
    assert np.max(pred_vessels) <= 1.0 and np.min(pred_vessels) >= 0.0
    assert np.max(true_vessels) == 1.0 and np.min(true_vessels) == 0.0
    assert np.max(masks) == 1.0 and np.min(masks) == 0.0
    assert pred_vessels.shape[0] == true_vessels.shape[0] and masks.shape[0] ==
        true_vessels.shape[0]
    assert pred_vessels.shape[1] == true_vessels.shape[1] and masks.shape[1] ==
        true_vessels.shape[1]
    assert pred_vessels.shape[2] == true_vessels.shape[2] and masks.shape[2] ==
        true_vessels.shape[2]

    if split_by_img:
        n = pred_vessels.shape[0]
        return (np.array([true_vessels[i, ...][masks[i, ...] == 1].flatten() for
            i in range(n)]),
                np.array([pred_vessels[i, ...][masks[i, ...] == 1].flatten() for
                    i in range(n)]))
    else:
        return true_vessels[masks == 1].flatten(), pred_vessels[masks == 1].
            flatten()


def remain_in_mask(imgs, masks):
    imgs[masks == 0] = 0
    return imgs


# noinspection PyPep8Naming
def AUC_ROC(true_vessel_arr, pred_vessel_arr):
    """
    Area under the ROC curve with x axis flipped
    ROC: Receiver operating characteristic
    """
    # roc_auc_score: sklearn function
    AUC_ROC_ = roc_auc_score(true_vessel_arr.flatten(), pred_vessel_arr.flatten
        ())
    return AUC_ROC_


# noinspection PyPep8Naming
def AUC_PR(true_vessel_arr, pred_vessel_arr):
    """
    Precision-recall curve: sklearn function
    auc: Area Under Curve, sklearn function
    """
    precision, recall, _ = precision_recall_curve(true_vessel_arr.flatten(),
```

```
                                                    pred_vessel_arr.flatten(),
                                                        pos_label=1)
    AUC_prec_rec = auc(recall, precision)
    return AUC_prec_rec


def threshold_by_f1(true_vessels, generated, masks, flatten=True, f1_score=False
    ):
    vessels_in_mask, generated_in_mask = pixel_values_in_mask(true_vessels,
        generated, masks)
    precision, recall, thresholds = precision_recall_curve(
        vessels_in_mask.flatten(), generated_in_mask.flatten(), pos_label=1)
    best_f1, best_threshold = best_f1_threshold(precision, recall, thresholds)

    pred_vessels_bin = np.zeros(generated.shape)
    pred_vessels_bin[generated >= best_threshold] = 1

    if flatten:
        if f1_score:
            return pred_vessels_bin[masks == 1].flatten(), best_f1
        else:
            return pred_vessels_bin[masks == 1].flatten()
    else:
        if f1_score:
            return pred_vessels_bin, best_f1
        else:
            return pred_vessels_bin


def best_f1_threshold(precision, recall, thresholds):
    best_f1, best_threshold = -1., None
    for index in range(len(precision)):
        curr_f1 = 2. * precision[index] * recall[index] / (precision[index] +
            recall[index])
        if best_f1 < curr_f1:
            best_f1 = curr_f1
            best_threshold = thresholds[index]

    return best_f1, best_threshold


def threshold_by_otsu(pred_vessels, masks, flatten=True):
    # cut by otsu threshold
    threshold = filters.threshold_otsu(pred_vessels[masks == 1])
    pred_vessels_bin = np.zeros(pred_vessels.shape)
    pred_vessels_bin[pred_vessels >= threshold] = 1

    if flatten:
        return pred_vessels_bin[masks == 1].flatten()
    else:
        return pred_vessels_bin


def dice_coefficient_in_train(true_vessel_arr, pred_vessel_arr):
    true_vessel_arr = true_vessel_arr.astype(np.bool)
    pred_vessel_arr = pred_vessel_arr.astype(np.bool)

    intersection = np.count_nonzero(true_vessel_arr & pred_vessel_arr)

    size1 = np.count_nonzero(true_vessel_arr)
    size2 = np.count_nonzero(pred_vessel_arr)
```

```
385
386     try:
387         dc = 2. * intersection / float(size1 + size2)
388     except ZeroDivisionError:
389         dc = 0.0
390
391     return dc
392
393
394 def misc_measures(true_vessel_arr, pred_vessel_arr):
395     cm = confusion_matrix(true_vessel_arr, pred_vessel_arr)
396     acc = 1. * (cm[0, 0] + cm[1, 1]) / np.sum(cm)
397     sensitivity = 1. * cm[1, 1] / (cm[1, 0] + cm[1, 1])
398     specificity = 1. * cm[0, 0] / (cm[0, 1] + cm[0, 0])
399     return acc, sensitivity, specificity
400
401
402 def difference_map(ori_vessel, pred_vessel, mask):
403     # ori_vessel : an RGB image
404     thresholded_vessel = threshold_by_f1(np.expand_dims(ori_vessel, axis=0),
405                                          np.expand_dims(pred_vessel, axis=0),
406                                          np.expand_dims(mask, axis=0), flatten=
                                               False)
407
408     thresholded_vessel = np.squeeze(thresholded_vessel, axis=0)
409     diff_map = np.zeros((ori_vessel.shape[0], ori_vessel.shape[1], 3))
410
411     # Green (overlapping)
412     diff_map[(ori_vessel == 1) & (thresholded_vessel == 1)] = (0, 255, 0)
413     # Red (false negative, missing in pred)
414     diff_map[(ori_vessel == 1) & (thresholded_vessel != 1)] = (255, 0, 0)
415     # Blue (false positive)
416     diff_map[(ori_vessel != 1) & (thresholded_vessel == 1)] = (0, 0, 255)
417
418     # compute dice coefficient for a given image
419     overlap = len(diff_map[(ori_vessel == 1) & (thresholded_vessel == 1)])
420     fn = len(diff_map[(ori_vessel == 1) & (thresholded_vessel != 1)])
421     fp = len(diff_map[(ori_vessel != 1) & (thresholded_vessel == 1)])
422
423     return diff_map, 2. * overlap / (2 * overlap + fn + fp)
424
425
426 def operating_pts_human_experts(gt_vessels, pred_vessels, masks):
427     gt_vessels_in_mask, pred_vessels_in_mask = pixel_values_in_mask(
428         gt_vessels, pred_vessels, masks, split_by_img=True)
429
430     n = gt_vessels_in_mask.shape[0]
431     op_pts_roc, op_pts_pr = [], []
432     for i in range(n):
433         cm = confusion_matrix(gt_vessels_in_mask[i], pred_vessels_in_mask[i])
434         fpr = 1 - 1. * cm[0, 0] / (cm[0, 1] + cm[0, 0])
435         tpr = 1. * cm[1, 1] / (cm[1, 0] + cm[1, 1])
436         prec = 1. * cm[1, 1] / (cm[0, 1] + cm[1, 1])
437         recall = tpr
438         op_pts_roc.append((fpr, tpr))
439         op_pts_pr.append((recall, prec))
440
441     return op_pts_roc, op_pts_pr
442
443
444 def misc_measures_evaluation(true_vessels, pred_vessels, masks):
```

```
    thresholded_vessel_arr, f1_score = threshold_by_f1(true_vessels,
        pred_vessels, masks, f1_score=True)
    true_vessel_arr = true_vessels[masks == 1].flatten()

    cm = confusion_matrix(true_vessel_arr, thresholded_vessel_arr)
    acc = 1. * (cm[0, 0] + cm[1, 1]) / np.sum(cm)
    sensitivity = 1. * cm[1, 1] / (cm[1, 0] + cm[1, 1])
    specificity = 1. * cm[0, 0] / (cm[0, 1] + cm[0, 0])
    return f1_score, acc, sensitivity, specificity


def dice_coefficient(true_vessels, pred_vessels, masks):
    thresholded_vessels = threshold_by_f1(true_vessels, pred_vessels, masks,
        flatten=False)

    true_vessels = true_vessels.astype(np.bool)
    thresholded_vessels = thresholded_vessels.astype(np.bool)

    intersection = np.count_nonzero(true_vessels & thresholded_vessels)

    size1 = np.count_nonzero(true_vessels)
    size2 = np.count_nonzero(thresholded_vessels)

    try:
        dc = 2. * intersection / float(size1 + size2)
    except ZeroDivisionError:
        dc = 0.0

    return dc


def save_obj(true_vessel_arr, pred_vessel_arr, auc_roc_file_name,
    auc_pr_file_name):
    fpr, tpr, _ = roc_curve(true_vessel_arr, pred_vessel_arr)  # roc_curve:
        sklearn function

    precision, recall, _ = precision_recall_curve(true_vessel_arr.flatten(),
                                                  pred_vessel_arr.flatten(),
                                                      pos_label=1)

    with open(auc_roc_file_name, 'wb') as f:
        pickle.dump({"fpr": fpr, "tpr": tpr}, f, pickle.HIGHEST_PROTOCOL)

    with open(auc_pr_file_name, 'wb') as f:
        pickle.dump({"precision": precision, "recall": recall}, f, pickle.
            HIGHEST_PROTOCOL)


def print_metrics(itr, kargs):
    print("*** Iteration {}  ====> ".format(itr))
    for name, value in kargs.items():
        print("{} : {:.6}, ".format(name, value))
    print("")
    sys.stdout.flush()


# noinspection PyPep8Naming
def plot_AUC_ROC(fprs, tprs, method_names, fig_dir, op_pts):
    # set font style
    font = {'family': 'serif'}
    matplotlib.rc('font', **font)
```

```
500
501     # sort the order of plots manually for eye-pleasing plots
502     colors = ['r', 'b', 'y', 'g', '#7e7e7e', 'm', 'c', 'k'] if len(fprs) == 8 \
503         else ['r', 'y', 'm', 'g', 'k']
504     indices = [7, 2, 5, 3, 4, 6, 1, 0] if len(fprs) == 8 else [4, 1, 2, 3, 0]
505
506     # print auc
507     print("****** ROC AUC ******")
508     print("CAVEAT : AUC of V-GAN with 8bit images might be lower than the
            floating point array "
509           "(check <home>/pretrained/auc_roc*.npy)")
510
511     for index in indices:
512         if method_names[index] != 'CRFs' and method_names[index] != '2nd_manual'
                :
513             print("{} : {:.4}".format(method_names[index], auc(fprs[index], tprs
                    [index])))
514
515     # plot results
516     for index in indices:
517         if method_names[index] == 'CRFs':
518             plt.plot(fprs[index], tprs[index], colors[index] + '*', label=
                    method_names[index].replace("_", " "))
519         elif method_names[index] == '2nd_manual':
520             plt.plot(fprs[index], tprs[index], colors[index] + '*', label='Human
                    ')
521         else:
522             plt.step(fprs[index], tprs[index], colors[index], where='post',
523                      label=method_names[index].replace("_", " "), linewidth=1.5)
524
525     # plot individual operation points
526     for op_pt in op_pts:
527         plt.plot(op_pt[0], op_pt[1], 'r.')
528
529     plt.title('ROC Curve')
530     plt.xlabel("1-Specificity")
531     plt.ylabel("Sensitivity")
532     plt.xlim(0, 0.3)
533     plt.ylim(0.7, 1.0)
534     plt.legend(loc="lower right")
535     plt.savefig(os.path.join(fig_dir, "ROC.png"))
536     plt.close()
537
538
539 # noinspection PyPep8Naming
540 def plot_AUC_PR(precisions, recalls, method_names, fig_dir, op_pts):
541     # set font style
542     font = {'family': 'serif'}
543     matplotlib.rc('font', **font)
544
545     # sort the order of plots manually for eye-pleasing plots
546     colors = ['r', 'b', 'y', 'g', '#7e7e7e', 'm', 'c', 'k'] if len(precisions)
            == 8 \
547         else ['r', 'y', 'm', 'g', 'k']
548     indices = [7, 2, 5, 3, 4, 6, 1, 0] if len(precisions) == 8 else [4, 1, 2, 3,
            0]
549
550     # print auc
551     print("****** Precision Recall AUC ******")
552     print("CAVEAT : AUC of V-GAN with 8bit images might be lower than the
            floating point array "
```

```
553             "(check <home>/pretrained/auc_pr*.npy)")
554
555     for index in indices:
556         if method_names[index] != 'CRFs' and method_names[index] != '2nd_manual'
                :
557             print("{} : {:.4}".format(method_names[index], auc(recalls[index],
                    precisions[index])))
558
559     # plot results
560     for index in indices:
561         if method_names[index] == 'CRFs':
562             plt.plot(recalls[index], precisions[index], colors[index] + '*',
563                      label=method_names[index].replace("_", " "))
564         elif method_names[index] == '2nd_manual':
565             plt.plot(recalls[index], precisions[index], colors[index] + '*',
                    label='Human')
566         else:
567             plt.step(recalls[index], precisions[index], colors[index], where='
                    post',
568                      label=method_names[index].replace("_", " "), linewidth=1.5)
569
570     # plot individual operation points
571     for op_pt in op_pts:
572         plt.plot(op_pt[0], op_pt[1], 'r.')
573
574     plt.title('Precision Recall Curve')
575     plt.xlabel("Recall")
576     plt.ylabel("Precision")
577     plt.xlim(0.5, 1.0)
578     plt.ylim(0.5, 1.0)
579     plt.legend(loc="lower left")
580     plt.savefig(os.path.join(fig_dir, "Precision_recall.png"))
581     plt.close()
```

Chapter 7

System Testing

The System testing done here with Test conditions and Test cases as follows:

7.1 Test Cases and Test Results

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
T01	Registration	The user must register before using detection feature	Registered successfully	Registered successfully and move on Login page
T02	Login	user is login in successfully before going to use other features	Login successfully	Login successfully and direct to dashboard
T03	Image Uploading	Upload image	Image uploading successfully and showing error when not having permission	Upload image successfully or show error if its is not in proper format
T04	Run model	Run model	RUN model in background and processing images	Call model and produce output image
T05	Test model	Test dataset are valid or not	Test the model and showing the accuracy of model	Show the accuracy
T06	Training	Taking input images dataset	Taking images dataset and shows the accuracy	Show output images and accuracy

7.2 Sample of a Test Case

Title: Login Page – Authenticate Successfully on Login Page
Description: A registered user should be able to successfully login at Web application and direct to the dashboard where they can access other features

Precondition: The user must already be registered with an Username and password.

Assumption: A supported browser is being used.

Test Steps:

1. Navigate to web application with address
2. In the 'Username' field, enter the username of the registered user.
3. Enter the password of the registered user
4. Click 'Log In'

Expected Result: A page displaying the dashboard of user where they can use other feature like detecting blood vessel and contribute images.

Actual Result: It should show the dashboard of the login user and on the dashboard it shows the functionality and specification to user to use those feature.

7.2.1 Software Quality Attributes

Our projects aim to attain high accuracy by model which have :

Availaibility: : our system should not been down when uploading image is happening and and model is running.

Scalablity: : In our project we can scaled upto as per our dataset has given to the system we can trained it and provide durable prediction for future

Reliability:: In this system we achieved high amout of relaibility to for access data

Prediction: :Prediction is the main highlight of the project which gives high precision in the image

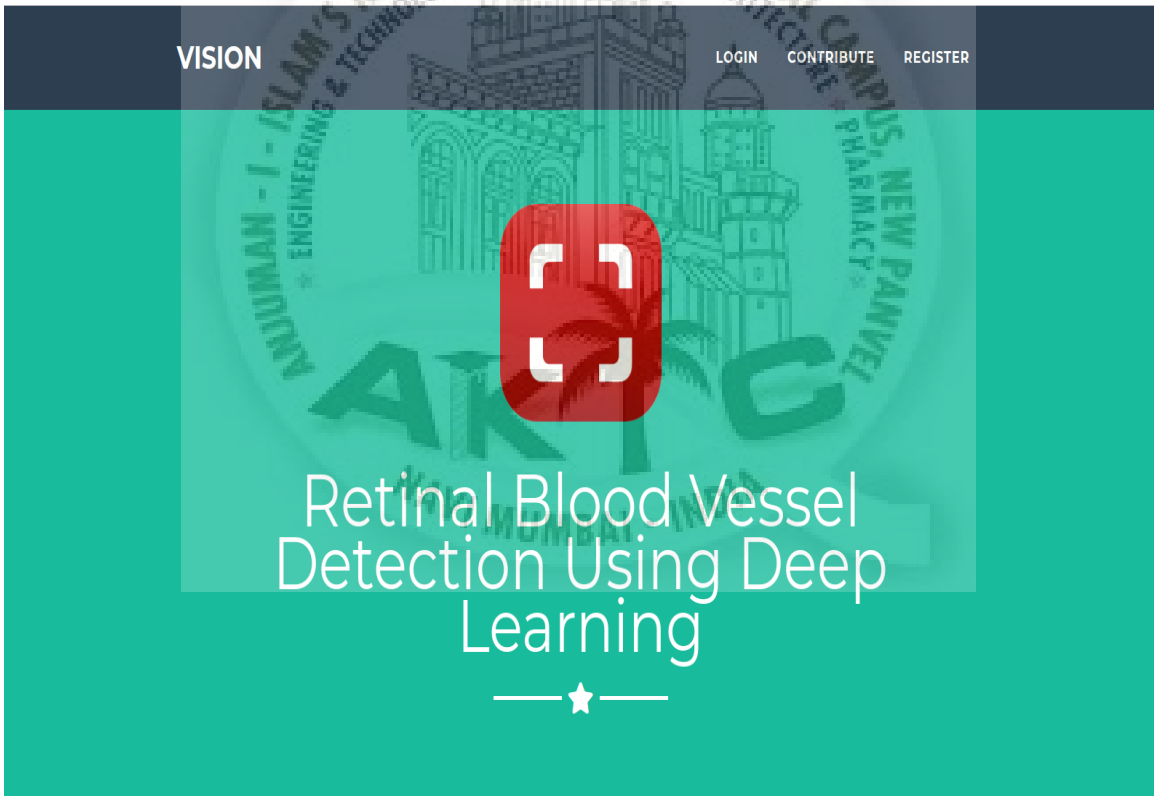
Clarity: : It give clarity in the retina which show cleanly blood vessel in sooth image format

Usability: : user can easily Use feature and software can be used by specified Doctor to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use.

Chapter 8

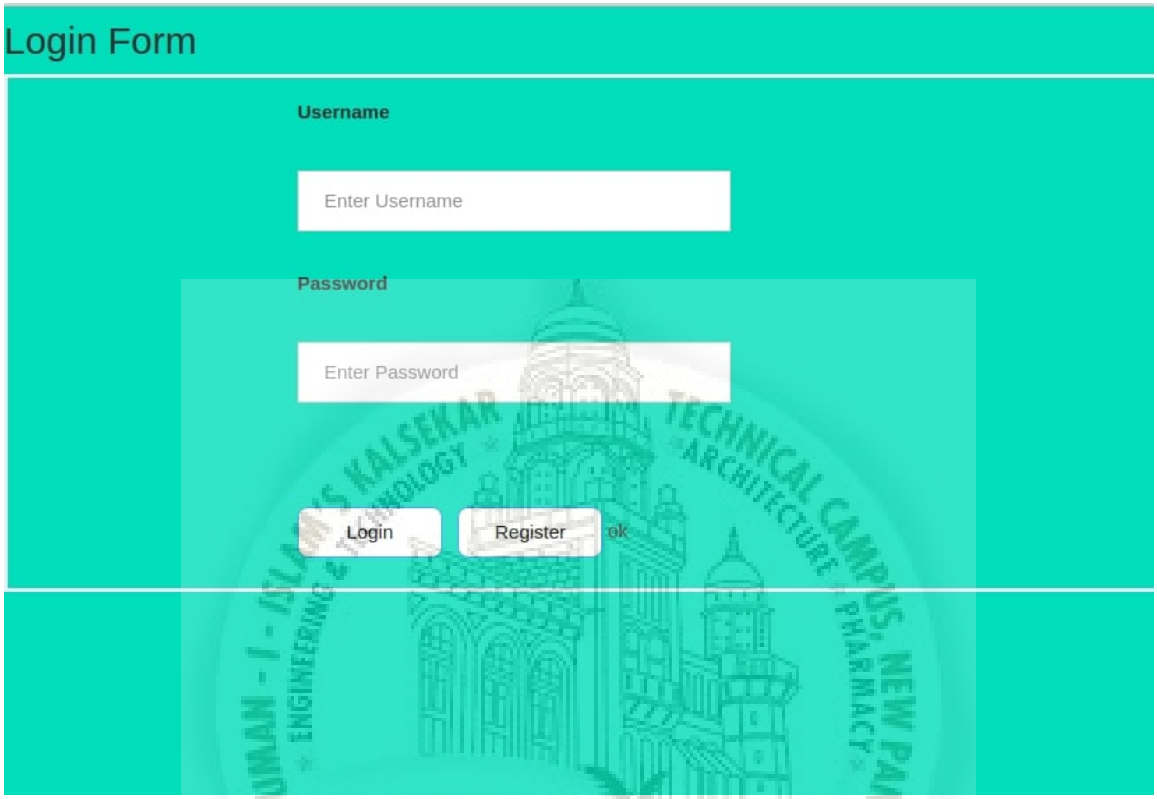
Screenshots of Project

8.1 Home Page

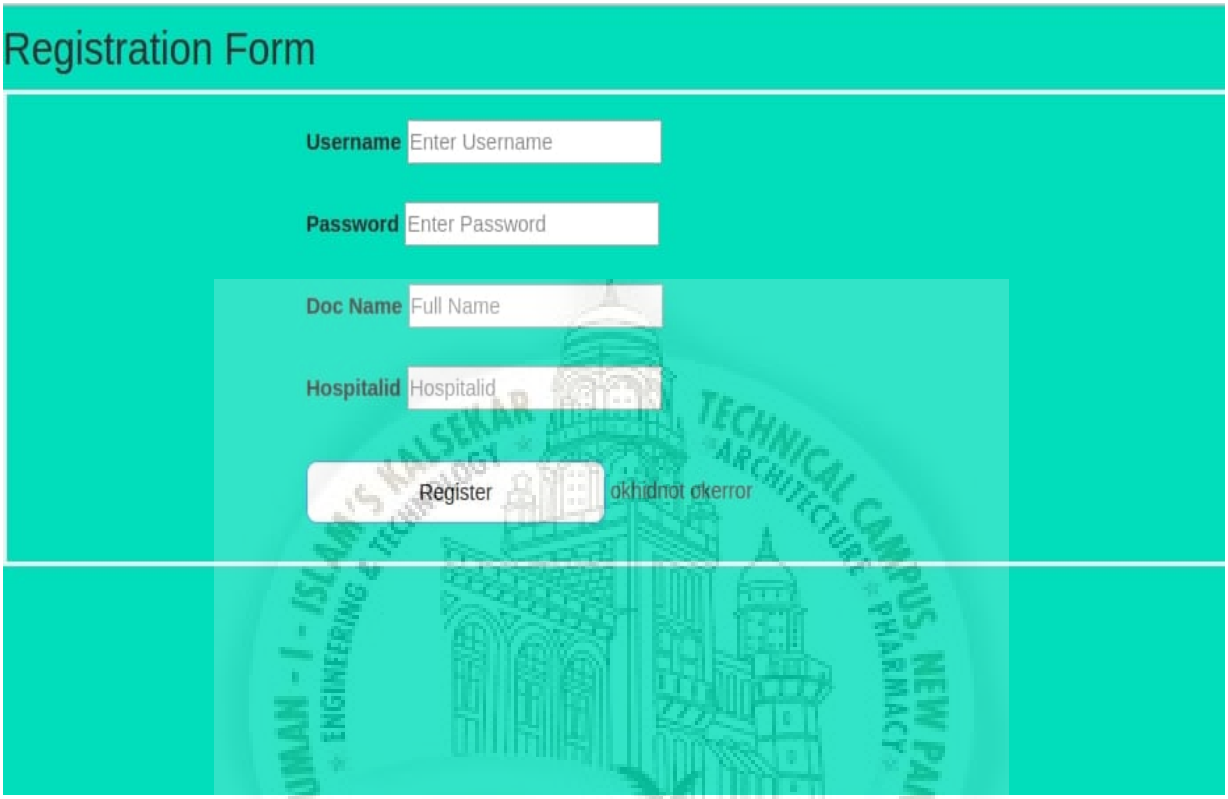


8.2 About Page

8.3 Login Page

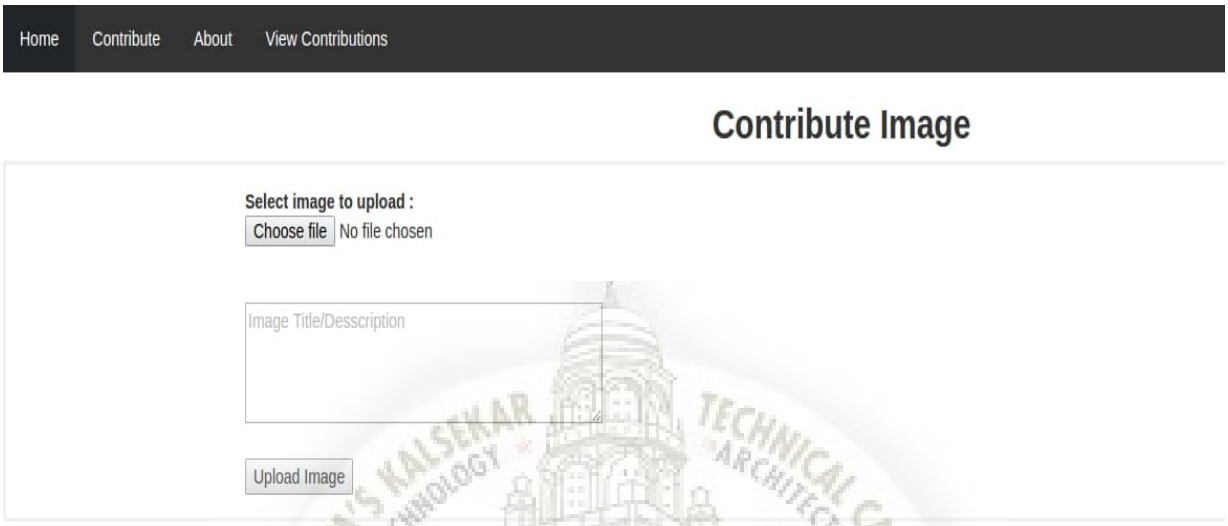


8.4 Registration Page



8.5 Upload Page

8.6 Contibute Page



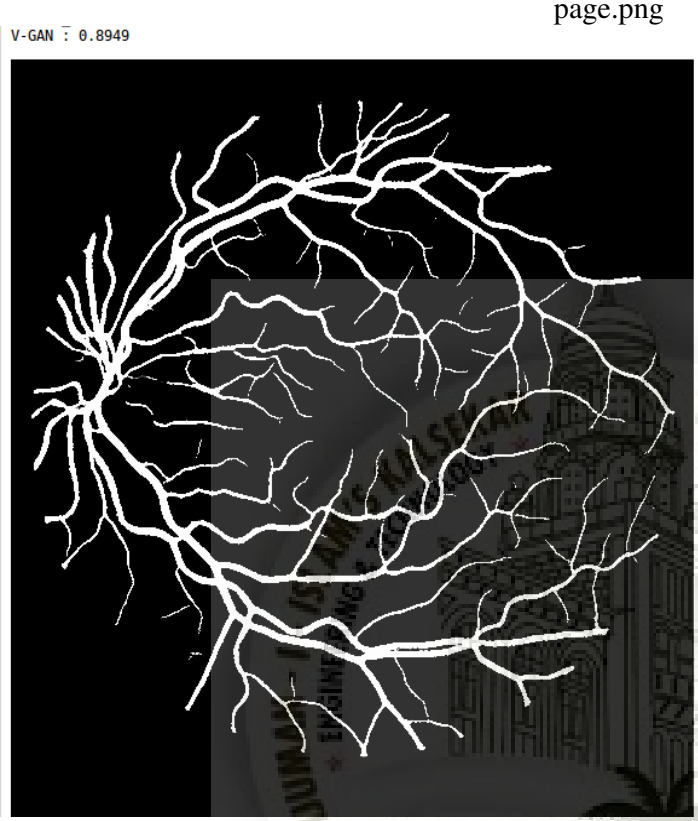
8.7 Runmodel Page

8.8 Accuracy Page

```
-- V-GAN --
dice coefficient : 0.8113
f1 score : 0.8115,
accuracy : 0.9520,
sensitivity : 0.8116,
specificity : 0.9725
```

8.9 Output Page

page.png



Chapter 9

Conclusion and Future Scope

9.1 Conclusion

Our proposed system will be able to enhance the the clarity and visibility of the retinal scans using supervised machine learning models(Neural Networks).Using GAN's results in much better predictability due to the use of U-net architecture. This method results in much results than any other algorithm. The goal is to provide consistent results which is big challenge in deep learning for varying types of input images.

9.2 Future Scope

- Better Designing of Neural Network could help to reduce the false positive generated but it will out of scope for this project
- Tweaking of neural network parameters can create a huge difference if we found out the appropriate hyper-parameter.

References

[1] Qiaoliang Li *Cross-modality Learning Approach For Vessel Segmentation in Retinal Images*

[2] Elisa Ricci and Renzo Perfetti *Retinal Blood Vessel Segmentation Using Line Operators and Support Vector Classification*

[3] Martina Melinscak,Pavle Prentasic and Sven Loncaric *Retinal Vessel Segmentation Using Deep Neural Networks*

Achievements

Conferences

1. *Detection of Retinal Blood Vessel using Deep Learning*; Sahil Jafar Khan Reshma,Shaikh Mohammed Yusuf Abdul Salam Mumtaz,Ansari Mohd Akram Sajid Ahmed Shamshad, AVALON , March 2019 (Venue : Terna Engineering College)

TERNA PUBLIC CHARITABLE TRUST'S
TERNA ENGINEERING COLLEGE
AN ISO 9001 : 2000 | NBA ACCREDITATION

terna
Knowledge to Empower

CERTIFICATE OF PARTICIPATION

This is to certify that

Yusuf Shaikh

of

A.I. Kalsekar Technical Campus

has participated in

Avalon 2019, A National Level
(Technical Paper Presentation / Project Competition)
conducted on 5th & 6th March, 2019
at Terna Engineering College, Nerul

Prof. D.M. Bavkar
Avalon co-ordinator

Dr. L.K. Ragha
Principal

TERNA PUBLIC CHARITABLE TRUST'S
TERNA ENGINEERING COLLEGE
AN ISO 9001 : 2000 | NBA ACCREDITATION

terna
Knowledge to Empower

CERTIFICATE OF PARTICIPATION

This is to certify that

Sahil Khan

of

A.I. Kalsekar Technical Campus

has participated in

Avalon 2019, A National Level
(Technical Paper Presentation / Project Competition)
conducted on 5th & 6th March, 2019
at **Terna Engineering College, Nerul**

AVALON

Prof. D.M. Bavkar
Avalon co-ordinator

Dr. L.K. Ragha
Principal

TERNA PUBLIC CHARITABLE TRUST'S
TERNA ENGINEERING COLLEGE
AN ISO 9001 : 2000 | NBA ACCREDITATION

terna
Knowledge to Empower

CERTIFICATE OF PARTICIPATION

This is to certify that

Akram Ansari

of

A.I. Kalsekar Technical Campus

has participated in

Avalon 2019, A National Level
(Technical Paper Presentation / Project Competition)
conducted on 5th & 6th March, 2019
at **Terna Engineering College, Nerul**

AVALON

Prof. D.M. Bavkar
Avalon co-ordinator

Dr. L.K. Ragha
Principal